Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE CARRERA

TÍTULO DEL TFC: Técnicas de procesado de datos de sónar de barrido lateral

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Carlos Agreda Ninot

DIRECTOR: Claudio Lo Iacono

FECHA: 28 de noviembre de 2007

Título: Técnicas de procesado de datos de sónar de barrido lateral

Autor: Carlos Agreda Ninot

Director: Claudio Lo Iacono

Fecha: 28 de noviembre de 2007

Resumen

En este documento se describe el desarrollo e implementación de técnicas para el procesado de datos de sónar de barrido lateral. Se presentan teóricamente las propiedades de los filtros implementados y se realiza una comparación con los resultados obtenidos.

El análisis de los datos geoespaciales de imágenes de sónar es uno de los recursos disponibles más importantes para el reconocimiento del fondo marino. El objetivo principal de este trabajo es implementar técnicas de realce que permitan obtener una mejor definición y realce de determinadas características como son los bordes y ciertos detalles visibles, elementos que contienen gran parte de la información de una imagen digital.

En concreto, los filtros de realce CS, WMMR-MED y Volterra son capaces de reducir en gran medida la presencia de ruido y suavizar regiones de la imagen que corresponden a zonas homogéneas del terreno escaneado.

El realce obtenido en las imágenes filtradas con dichas técnicas facilita posteriores procesos como la clasificación de imágenes o el análisis de determinadas texturas de la superficie del lecho marino.

Las técnicas de realce han sido implementadas en respectivos módulos preparados para su integración en el software de código abierto OpenEV, una aplicación destinada al procesado de datos geoespaciales. Para ello se ha utilizado el lenguaje de programación Python, además de diversas funcionalidades que ofrece la distribución FWTools para el desarrollo de aplicaciones destinadas a OpenEV.

Finalmente, los resultados obtenidos en las pruebas de filtrado han servido para caracterizar la elección de parámetros en función de los objetivos iniciales del usuario, y para considerar una futura ampliación de las opciones ofrecidas por los módulos.

Title: **Side scan sonar data processing techniques**

Author: Carlos Agreda Ninot

Director: Claudio Lo Iacono

Date: November, 28th 2007

Overview

In this document is described the development and implementation of side scan sonar data processing techniques. The properties of the implemented filters are theoretically showed and compared with reached results.

Sonar image geospatial data analysis is one of the most important available resources to seafloor recognition. The main goal of this project is to implement enhancement techniques that allow to obtain a better image definition and to enhance certain characteristics as edges or some visible details, which have a great amount of information about digital images.

More specifically, CS filter, WMMR-MED filter and Volterra filter are able to reduce significantly noise appearance and to smooth image parts that belong to homogeneous regions of the scanned surface.

The achieved enhancement about filtered images using these techniques make easier the subsequent process as image classification or seafloor surface texture analysis, for example.

These enhancement techniques have been implemented in their own modules, which are prepared in order to their integration in OpenEV software, an open source application focused to geospatial data processing. Python language programming is used for that purpose, besides several functionalities offered by FWTools distribution to OpenEV applications development.

To conclude, the achieved results of filtering tests have been useful to determine the parameters choice with regard to the initial aim user and to take account of a possible future extension of options' module.

ÍNDICE

INTRODUCCIÓN

Las técnicas de procesado de datos de sónar de barrido lateral permiten extraer determinadas características de las imágenes obtenidas. Gracias a esto se obtiene valiosa información que facilita un análisis más profundo y una mejor clasificación de las imágenes.

El objetivo de este trabajo es analizar tres filtros espaciales de realce, e implementarlos en módulos destinados a la aplicación OpenEV. Los filtros CS, WMMR-MED y Volterra se caracterizan por aumentar el realce y la definición de las imágenes, además de reducir en gran medida la presencia de ruido. Los filtros proporcionan imágenes con más nitidez y mejor calidad visual de cara al ojo humano.

En el capítulo 1, *'Captación y características de imágenes de sónar'*, se hace referencia al proceso de captación de una imagen de sónar de barrido lateral y se presentan las características de las imágenes de sónar. Se introduce la problemática de la aparición del ruido, los tipos de ruido más influyentes en imágenes digitales y las causas de su origen. También se tratan las alteraciones que producen ciertos tipos de ruido en la apariencia visual de una imagen digital.

En el capítulo 2, *'Realce de imágenes digitales'*, se introducen los tipos de filtrado existentes para el realce de imágenes de sónar. Dado que los filtros a implementar en este trabajo son filtros espaciales, se pone énfasis en el filtrado espacial, así como en los filtros no lineales. Además se incluye un subapartado donde se muestran diferentes tipos de estructura de ventana y se analiza el utilizado en los filtros.

En el capítulo 3, *'Filtros implementados'*, se analizan los filtros CS, WMMR-MED y Volterra. Cada filtro es definido, se enumeran sus propiedades y se presentan sus características. En el subapartado de características también se trata teóricamente los efectos que tiene una determinada elección de parámetros, y se analiza la interfaz gráfica de usuario de cada módulo implementado.

En el capítulo 4, *'Software'*, se presentan las aplicaciones usadas en la implementación de los filtros, sus características y funcionamiento general.
Se presta atención especial al programa OpenEV, que es el software de código abierto al que van destinados los módulos de realce de este trabajo. También se dedica un apartado al lenguaje de programación Python que es el utilizado en el código de los filtros.

En el capítulo 5, *'Resultados de las pruebas de filtrado'*, se muestran los resultados obtenidos en relación a las imágenes filtradas seleccionadas. La comparación considera una muestra de imágenes con diferentes valores de parámetros, con el objetivo de valorar el efecto de una determinada elección.

Por último, se exponen los objetivos del trabajo y cómo se ha desarrollado, incluyendo su organización y la problemática encontrada durante su proceso. También se incluye un subapartado referido al impacto medioambiental que éste puede suponer, y se finaliza con las conclusiones extraídas una vez finalizado el trabajo.

CAPÍTULO 1. CAPTACIÓN Y CARACTERÍSTICAS DE IMÁGENES DE SÓNAR

En los siguientes apartados se presentan los instrumentos utilizados para la captación de las imágenes sobre las que actúan las técnicas de procesado que se implementan en este trabajo. En primer lugar se introduce el concepto de sónar, para luego tratar el sónar de barrido lateral.

A continuación se analizan las características de las imágenes de sónar, y las causas y efectos de la aparición de ruido en ellas.

1.1. Sónar

El sónar ("Sound Navigation and Ranging") es el equipo, medio o propiedades que usa la propagación del sonido en el agua para determinar la ubicación y características de objetos en el fondo marino, así como para comunicarse o detectar embarcaciones.

Dependiendo del funcionamiento y uso del sónar se pueden encontrar dos tipos:

- *Sónar activo:* Instrumento que detecta objetos bajo el agua empleando el eco devuelto por el objeto en el cual inciden las ondas acústicas emitidas por un transductor.

- *Sónar pasivo:* Instrumento que se limita a escuchar el sonido proveniente de los objetos sumergidos.

En general, se suelen combinar ambos tipos de sónar para detectar y analizar objetos del lecho marino de forma más eficaz.

A continuación, el apartado centra la atención en el sónar activo ya que es el utilizado en la captación de las imágenes de sónar empleadas en este trabajo. Este tipo de sónar emite mediante un transductor un tren de ondas acústicas con una determinada potencia al agua, con el objetivo de que incida sobre un objeto y éste refleje parte de ellas. El eco recibido por el transductor permite detectar objetos sumergidos y obtener información sobre su distancia, dirección y movimiento.

En el caso de recibir un eco proveniente de ondas acústicas emitidas por el transductor, el receptor centra la escucha en un pequeño ancho de banda centrado en la frecuencia transmitida, ya que la señal recibida llega un poco desplazada. Además el eco recibido corresponde a una parte de la energía emitida, la cual ha perdido intensidad debido a las pérdidas que la señal sufre a lo largo del recorrido. En el apartado 1.5 se amplía información a cerca de algunas de las causas que provocan la pérdida de potencia de la señal.

Además los sistemas de sónar activo actuales también pueden funcionar como sónar pasivo con ciertas limitaciones debidas a las diferencias existentes en las características de los transductores de ambos sistemas.

1.2. Sónar de barrido lateral

El sónar de barrido lateral o *Side Scan Sonar* (SSS) es un implemento electrónico que utiliza un elemento sumergido (por lo general un torpedo) arrastrado por un buque, el cual envía una señal de sónar hacia el fondo y recibe el eco del mismo (ver [24]). Posteriormente el eco se procesa y se traduce la señal en una imagen digitalizada. Simultáneamente se tiene la posición del torpedo controlada por GPS, lo que proporciona la posición exacta de la imagen del fondo marino captada.

Al ir remolcado se suelen introducir errores en la posición de la imagen, los cuales pueden ser reducidos realizando otra pasada o haciendo uso de respondedores. También se producen limitaciones debidas al equipo de captación utilizado, siendo una de las principales la velocidad de arranque requerida para lograr suficientes pulsos (ping) para detectar correctamente el accidente.

El sónar de barrido lateral tiene cuatro funciones a destacar:

- La detección de naufragios y obstrucciones entre las líneas de sondaje.
- La detección de otros accidentes del fondo marino, incluso los de muy pequeño tamaño.
- La agrupación de los datos de clasificación del fondo marino, combinando el conocimiento de su textura y muestras del fondo.
- La identificación de áreas dinámicas del fondo marino, realizando un seguimiento sobre ellas para garantizar la seguridad de la navegación.

La cantidad y calidad de la información que es capaz de proporcionar el sónar de barrido lateral lo ha convertido en un instrumento de uso extendido a la hora de realizar mediciones y escaneos del fondo marino. Estos datos combinados con muestras del fondo y el análisis de los contornos de la profundidad son unas herramientas ideales para la clasificación del fondo marino.

1.3. Imágenes de sónar

Las imágenes de sónar son un importante recurso de información para el análisis de las características del fondo marino.

Las imágenes obtenidas mediante sónar se caracterizan por una presencia generalmente extensa del ruido y por una falta de definición de los bordes. La aparición de ruido puede ser debida a diversas causas, las cuales se pueden

agrupar en causas debido al error introducido por el equipo de captación y las debidas a las condiciones del medio físico en el momento de la captura.

Las imágenes detectadas remotamente en general, y las imágenes captadas mediante sónar en concreto, son descritas principalmente por sus propiedades de textura y tonales. En el caso de las imágenes de sónar, el tono corresponde a la cantidad de energía recibida por cada punto, denominado *backscattering*, y se expresa como niveles de gris.

El *backscattering* se ve afectado por diversos motivos, como ya se ha mencionado. La causa más influyente es la geometría del transductor, las características morfológicas de la superficie (rugosidad a escala microscópica) y por su naturaleza intrínseca (composición, densidad, importancia relativa del volumen y la resonancia de la superficie). La mayoría de los procesos físicos y sus manifestaciones en la superficie no se pueden describir únicamente con sus propiedades tonales, lo que hace necesario el análisis de la textura del fondo marino.

Las propiedades de textura corresponden a la organización espacial de los niveles de grises entre píxeles vecinos. Las texturas se pueden describir intuitivamente como suaves o ásperas, si corresponden a un terreno a pequeña o a gran escala, aleatorias u organizadas, etc. El tipo de textura de la superficie de una región captada se puede diferenciar según la homogeneidad en el nivel de gris de los píxeles, por el grado de granulado de la región o por el tipo de detalles que se pueden apreciar en él.

1.4. Ruido en imágenes de sónar

La aparición de ruido en imágenes de sónar es un hecho habitual que modifica la información real de la imagen y produce datos arbitrarios que se confunden con los originales. En (**Fig. 1.1**) se muestra una imagen de sónar con una importante presencia de ruido, lo que dificulta procesos como la detección de bordes o el realce de características determinadas. Así, se hace imprescindible el conocimiento de los tipos de ruido más frecuentes en imágenes digitales para un correcto análisis de las mismas.

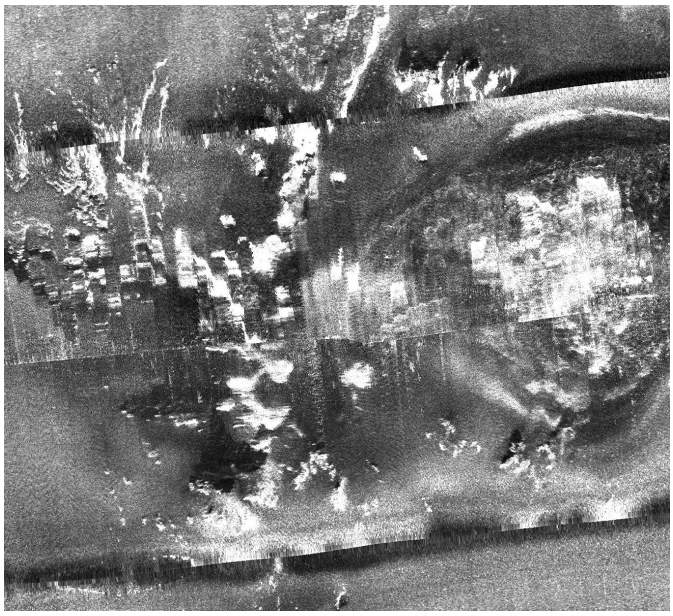

Fig. 1.1 Imagen de sónar del fondo marino de las costas de Almería con una presencia generalizada de ruido.

Se entiende por ruido en imágenes digitales cualquier valor de un píxel de una imagen que no se corresponda exactamente con la realidad. En general, el ruido se debe al equipo electrónico utilizado en la captación de las imágenes y al ruido añadido en su transmisión.

1.4.1. Ruido blanco gaussiano

Se entiende por ruido blanco gaussiano una señal arbitraria con una densidad espectral constante y una amplitud que sigue una distribución de Gauss, por lo que su espectro de energía es constante para todas las frecuencias (ver [14]).

El ruido blanco gaussiano tiene un efecto general en las imágenes, siendo éste una suma o resta de un determinado valor al nivel de gris real, independientemente del valor original del mismo. Por lo tanto para un píxel cualquiera por cada captura de la misma imagen se obtiene un valor alterado en cierta medida con respecto al original.

Este tipo de ruido puede ser debido a diversas fuentes naturales como vibraciones o al ruido de disparo que es comentado en el apartado 1.4.

1.4.2. Ruido impulsivo

El ruido impulsivo se caracteriza por aparición de píxeles con valores arbitrarios, generalmente con valores extremos y alejados de los de sus vecinos más cercanos.

La distribución que sigue viene dada por

$$g(x,y) = \begin{cases} 0 & \text{si} \quad r(x,y) < p/2 \\ L\text{-}1 & \text{si} \quad p/2 \le r(x,y) < p \\ f(x,y) & \text{si} \quad r(x,y) \ge p \end{cases} \qquad \textbf{(1.1)}$$

donde $r(x,y)$ es un número aleatorio con distribución uniforme [0,1) y p es la probabilidad de ocurrencia del ruido aleatorio, o lo que es lo mismo, el porcentaje de puntos de la imagen que se verían afectados por el ruido impulsivo.

El conjunto de píxeles afectados toman valores extremos, siendo una buena parte de ellos los valores 0 ó 255.

El ruido impulsivo tiene una notable influencia en la calidad visual de la imagen, debido a la diferencia de intensidades entre el píxel ruidoso y el conjunto de píxeles vecinos. Esto puede deteriorar bordes y detalles, y distorsionar el realce de ciertas características de la imagen.

1.4.3. Ruido grano

El ruido grano, también denominado ruido estructural, es una variación de la señal reflejada en una región no homogénea y produce una apariencia de imagen granulada (ver [19] y [28]). Éste provoca cambios en las estadísticas espaciales de la región y una reducción en el contraste, lo que dificulta el posterior análisis y clasificación de las texturas de la superficie del terreno captado en la imagen.

El ruido grano se debe a los ecos que producen cierto tipo de estructuras y que se presentan en forma de ruido coherente, es decir, dentro de la misma banda de frecuencias que las señales de ultrasonidos asociadas a interfases o defectos.

Los materiales que producen este efecto son de diverso origen, encontrándose entre ellos materiales metálicos o compuestos. Estos poseen una estructura interna formada por pequeños reflectores que no son resolubles por la longitud de onda del haz ultrasónico que incide.

1.5. Causas del ruido en imágenes de sónar

Las imágenes de sónar se ven afectadas por diferentes tipos de ruido [11], los cuales se pueden clasificar en ruido ambiental (movimientos parásitos, fluctuaciones de la velocidad del agua e interferencias multidireccionales), y en ruido relacionado con el método físico de adquisición de la información (propagación de la onda, ruido speckle y la longitud del tren de onda).

A continuación se enumeran algunas causas de la aparición del ruido ambiental:

- *Movimientos parásitos:* La señal adquirida sufre los efectos del movimiento del barco que remonta al sónar, los cuales se transmiten por el cable y pueden inducir en errores de precisión en la altura del objeto sumergido.

- *Fluctuación de la velocidad:* Las trayectorias de las ondas acústicas son curvas debido a que el índice de propagación en el agua no es una constante, sino que varía en función de diversos parámetros del medio, entre ellos la presión.

- *Interferencias multidireccionales:* La aparición de señales parásitas con direcciones de reflexión diferentes a la de la señal original, contribuye a formar ruido en el frente de onda de la señal principal. Este hecho constituye una importante fuente de ruido y puede perturbar la medición de la dirección de la señal recibida.

El ruido físico se puede presentar en una imagen debido a las siguientes causas:

- *Puntos de referencia no correlacionados:* El valor del nivel de gris medido perteneciente a un píxel cualquiera de la imagen de sónar se obtiene por la suma de ecos de señales generados aleatoreamente contenidos en la celda de resolución. Se crea el denominado ruido grano, una importante fuente de ruido.

- *Longitud del tren de onda:* Esta fuente de ruido puede proceder de un efecto óptico y provoca una degradación progresiva del contraste desde el centro hacia fuera.

- *Pérdidas de propagación:* Los efectos de la propagación son importantes a frecuencias altas debido a la resonancia de las moléculas del agua del mar. La ecuación del sónar es:

Existen también ruidos de tipo electrónico. En relación a imágenes de sónar se destaca:

- *Ruido de disparo:* Tipo de ruido electrónico que se origina por las fluctuaciones de los fotones detectados, cuando éste es un número finito suficientemente pequeño

CAPÍTULO 2. REALCE DE IMÁGENES DIGITALES

Los algoritmos de realce de imágenes son aplicados a imágenes captadas remotamente con el objetivo de mejorar la apariencia de una imagen para el análisis visual humano u ocasionalmente para un posterior análisis por computadora.

Existen técnicas de realce tanto en el dominio espacial, como en el frecuencial (ver Anexo B).

En general el realce mediante filtros espaciales tiene un uso más extendido que los filtros basados en transformadas de Fourier debido a su facilidad de implementación y a su velocidad de operación. No obstante, éstos últimos son capaces de solucionar problemas que mediante técnicas espaciales no son capaces de resolver.

2.1. Filtrado espacial

El filtrado espacial es la operación que se aplica a una imagen para resaltar o atenuar detalles espaciales con la intención de mejorar la calidad de la imagen respecto a las capacidades del sistema visual humano o de facilitar en su caso, un posterior procesamiento.

El filtrado espacial es una operación local ya que modifica el valor de cada píxel de acuerdo con los valores de intensidad de los píxeles que lo rodean. Se trata de transformar los niveles de gris originales de manera que se parezcan o diferencien más de los correspondientes píxeles vecinos.

Los filtros espaciales son analizados en el siguiente aparatado.

2.3. Filtros espaciales

La frecuencia espacial es un parámetro característico de las imágenes detectadas remotamente, el cual se define como el número de cambios en el valor del brillo por unidad de distancia para cualquier parte de una imagen. Dada un área determinada, el grado de variación del valor del brillo es el que determina si una zona es de baja frecuencia, si tiene muy pocos cambios en el valor del brillo, o una zona de alta frecuencia si los valores del brillo cambian en gran medida sobre distancias cortas.
La información útil a extraer por lo tanto es el brillo de los píxeles locales y su variación entre píxeles vecinos.

Los filtros espaciales se basan en el procesado de ésta información respecto a un área determinada, modificando los valores de brillo de la imagen y proporcionando una imagen saliente realzada o por el contrario, atenuada.

Una posible clasificación de los filtros espaciales se basa en su linealidad, distinguiendo entre filtros lineales y filtros no lineales. Así mismo, los primeros se pueden clasificar según las frecuencias que dejen pasar, sirviendo a modo de ejemplo los siguientes tipos: los filtros paso alto atenúan o eliminan las componentes de baja frecuencia, realzando zonas de alto contenido frecuencial tales como bordes; los filtro paso bajo atenúan o eliminan las componentes de alta frecuencia, suavizando por lo tanto la imagen; los filtros paso banda eliminan zonas elegidas de frecuencias intermedias.

Los filtros lineales se basan en el uso de máscaras que recorren toda la imagen centrando las operaciones sobre los píxeles encuadrados en el área de la imagen original que coincide con la máscara, siendo el resultado una suma de convolución entre los píxeles originales y los diferentes coeficientes de las máscaras, dándole el nombre de máscaras de convolución.

Del otro lado, los filtros no lineales basan su operación directamente en los valores de los píxeles del entorno al píxel central. Uno de los ejemplos de este tipo de filtros más habituales es el filtro mediana, el cual tiene un efecto de difuminado de la imagen, permitiendo eliminar gran cantidad de componentes de ruido de forma eficaz.

También pertenecen a este grupo los filtros CS, WMMR-MED y Volterra tratados en este documento, siendo todos ellos filtros de realce, por lo que suprimen gran cantidad de ruido a la vez que realzan los bordes, dando a la imagen una mejor definición. Todos estos filtros son conocidos como filtros de rango.

Otro tipo de clasificación sería en base a su finalidad, distinguiendo de este modo los filtros de realce que permiten eliminar zonas borrosas y definir zonas de alto contenido frecuencial, filtros de suavizado para difuminar la imagen o los filtros diferenciales que se componen de varios tipos de máscaras y son usados para la detección de bordes. De esta clasificación, el tipo que engloba los tres filtros a analizar es el de filtros de realce de la imagen.

Los filtros de realce tienen como objetivo resaltar los detalles más finos de la imagen, tales como bordes, e intensificar aquellos que se encuentren difuminados por el ruido, eliminando en cierto grado la borrosidad que hubiera provocado el ruido en dichas zonas. A cambio, el uso de este tipo de filtro implica una pérdida de pequeños detalles y una apariencia global de la imagen resultante algo deteriorada.

Los filtros de realce lineales se basan en el uso de máscaras de convolución, siendo la salida del filtro el resultado de una operación directa entre las muestras que contiene en ese momento la ventana del filtro. El filtro de realce más habitual es el filtro paso alto, el cual elimina las componentes bajas de frecuencia dejando pasar el resto.

Por lo tanto, las características de los filtros de realce lineales no permiten obtener los resultados deseados en el procesado de una imagen adquirida remotamente, debido a la gran cantidad de ruido existente en ellas. La falta de

propiedades que contrarresten los efectos del ruido, en especial las componentes impulsivas, obliga a buscar alternativas para el filtrado de este tipo de imágenes.

2.4. Filtros de realce no lineales

El estudio de filtros de realce no lineales es promovido por la importancia de la información que constituyen los contornos en el estudio de imágenes digitales obtenidas remotamente. La obtención de unos contornos bien definidos es una fuente de información vital a la hora de analizar una imagen, siendo a la vez uno de los mayores problemas a la hora de tratar con imágenes obtenidas por sónar. La importancia de conseguir unos filtros que realcen los bordes de forma óptima viene dada por la cantidad de información que se puede extraer a partir de los ellos, tales como la forma, tamaño, textura y posición del objeto que delimitan.

A la hora de procesar imágenes originales obtenidas mediante sónar se añaden los inconvenientes que el proceso de captación de la mismas ha originado, la superposición de ruido blanco gaussiano así como la aparición de componentes de ruido impulsivo, difuminando en gran medida los bordes, y según las circunstancias generando estructuras y detalles arbitrarios.

El filtrado de realce no lineal es adecuado si la imagen a procesar es borrosa y tiene bordes poco marcados, dado que este tipo de filtros permite eliminar gran cantidad de ruido al mismo tiempo que consiguen un buen realce de los bordes originales. Además de estas cualidades, los tres filtros pueden diferenciar los bordes de la imagen original de los aparecidos a consecuencia del ruido.

Por lo tanto, el filtrado CS, WMMR-MED y Volterra se consideran soluciones ideales para el realce de imágenes adquiridas remotamente, o para un prefiltrado si el objetivo es preparar la imagen para un posterior tratamiento, como así sería la detección de contornos. Este prefiltrado es básico para que el posterior tratamiento de la imagen obtenga un resultado óptimo, ya que la mayoría no funcionan correctamente en imágenes con un alto grado de ruido.

Para eliminar estos efectos no deseados en las imágenes originales la mejor herramienta son los filtros de realce no lineales, los cuales suavizan y eliminan el nivel de ruido, y realzan y delimitan los bordes. Además no resaltan falsos bordes creados a partir de ruido, sino que delimitan zonas con un mismo gradiente de forma suave y uniforme, eliminando el ruido impulsivo situado en determinados píxeles con valores extremos.

A continuación se extiende una lista de términos que resultan básicos para la definición de los filtros analizados:

- *Segmento constante:* Corresponde a un segmento de N+1 o más valores iguales.

- *Borde perfecto:* Dos segmentos no iguales separados por al menos una muestra de valor contenido entre los dos segmentos.

- *Borde imperfecto:* Es un segmento compuesto de dos segmentos constantes no iguales separados por un segmento de longitud menor a *N*+1, limitada por los valores de las regiones constantes.

- *OS filter (Order statistic filter):* Filtro cuya salida es la suma algebraicamente ordenada por pesos de los valores de la ventana.

- *GOS filter (General order statistic filter):* Filtro cuya salida es la suma por pesos de valores ordenados, y donde los valores son seleccionados por un proceso de ordenación diferente a una simple ordenación algebraica.

- *Gradiente:* Vector de dirección perpendicular a las curvas de nivel de la superficie y sentido el de crecimiento de la función.

2.2. Filtrado frecuencial

El filtrado en el dominio frecuencial incluye técnicas que están basadas en la modificación de la transformada de Fourier de la imagen (ver [1] y [2]). El proceso seguido por las técnicas se inicio con el cálculo de la transformada de Fourier de la imagen, a continuación se multiplica por la función de transferencia del filtro, y para finalizar se calcula la transformada inversa para conseguir la imagen transformada.

Los filtros frecuenciales basan su funcionamiento en la eliminación de las bandas de frecuencias que no interesan, dejando pasar solamente las que contienen información útil para su propósito. Los filtros paso bajo atenúan o suprimen las componentes de alta frecuencia en el dominio de Fourier, mientras dejan sin alterar las bajas. El efecto de un filtro paso bajo es el de difuminar la imagen, debido a que las altas frecuencias son características de bordes, curvas y detalles en la imagen. Así mismo, los filtros paso alto atenúan o eliminan las bajas frecuencias, que son las responsables de las pequeñas variaciones de las características de una imagen, como son el contraste global y la intensidad media. El filtro paso alto reduce estas características, y permite por el contrario, el realce de bordes y la aparición de detalles en la imagen resultante.

2.5. Estructuras de ventana deslizante

El tipo de estructura de ventana empleada en el algoritmo es uno de los factores más influyentes en el resultado final del filtrado (ver [5]).

Existen diversos tipos de ventana, cada uno de ellos posee ciertas ventajas y desventajas respecto al resto y por lo tanto, hay un tipo de estructura de ventana apropiado para cada finalidad. Entre las estructuras de uso más habitual nos encontramos las mostradas en (**Fig. 2.1**).

La primera de ellas corresponde a una ventana cuadrada, mostrada en (**Fig. 2.1 (a)**), la cual posee mayor capacidad de realce que el resto de estructuras al contar con un número mayor de valores, siendo el tipo de ventana que se utiliza de forma más generalizada en la implementación de filtros. En (**Fig. 2.1 (b)**) se muestra una ventana rectangular, la cual se caracteriza por una cantidad de realce diferente horizontal o verticalmente. La diferencia de realce en ambas direcciones viene determinada por la cantidad de muestras correspondientes a cada dirección que la ventana adquiere, consiguiendo un mayor realce en la dirección en la que la ventana tiene más longitud.

En (**Fig. 2.1 (c)**) se muestra una estructura circular, de similar características que la ventana cuadrada pero con una menor capacidad de realce debido al menor número de muestras.
En (**Fig. 2.1 (d)**) se presenta una ventana unidimensional que podría ser utilizada para una señal bidimensional. El filtrado con este tipo de estructura produce por lo general realces unidireccionales en el eje de la ventana del filtro.

Una ventana con estructura en forma de cruz es mostrada en (**Fig. 2.1 (e)**), la cual realza los contornos de igual forma en ambos ejes longitudinales. Los efectos son similares a los que se consiguen con una ventana cuadrada pero con una cuarta parte de la potencia a igual tamaño, debido a la menor cantidad de muestras que contiene. Es una excelente opción a utilizar en lugar de la ventana cuadrada en condiciones de poco ruido, ya que al procesar menos muestras por ventana aumenta la rapidez del algoritmo, resultando un proceso más rápido y con resultados de realce similares.

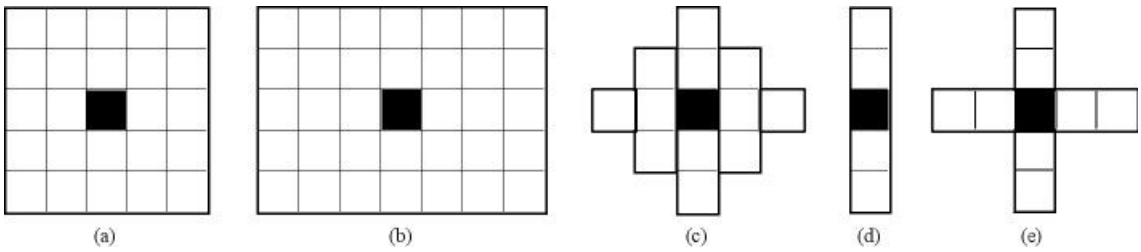

Fig. 2.1 Estructuras de ventana bidimensionales. **(a)** Cuadrada. **(b**) Rectangular. **(c)** Circular. **(d)** Unidimensional. **(e)** Cruz

Dado que las imágenes captadas por sónar contienen una gran cantidad de ruido, es preferible utilizar una estructura de ventana cuadrado para el filtro, ya que contiene un número mayor de muestras que el resto, con lo que se atenúa con mayor eficacia las componentes de ruido. Las muestras redundantes quedan por tanto difuminadas en la secuencia, posibilitando que en casos con insensibilidad a los valores extremos, permita el uso de valores altos de los parámetros *J* y *N,* que son analizados en el capítulo 3.

La estructura cuadrada permite realzar la imagen resultante en ambas direcciones longitudinales con la misma potencia, de forma que se mantiene una proporcionalidad en el realce en toda ella.

La ventana de filtro cuadrada es, por lo tanto, la elegida para implementar en los siguientes filtros de realce, pudiendo elegir entre diversos tamaños.

Las ventanas ofrecidas al usuario en los módulos de los filtros implementados están comprendidas entre un tamaño de 3x3 hasta 15x15, siendo siempre valores impares ya que esto posibilita que la ventana esté centrada en la muestra entrante a filtrar.

CAPÍTULO 3. FILTROS IMPLEMENTADOS

Los siguientes apartados se centran en el análisis de los filtros CS, WMMR-MED y Volterra, por lo que se hace necesario definir un escenario:

Se considera una secuencia discreta *d*-dimensional {x(n)} donde el índice n = [n_1,n_2,⋯,n_d] y una ventana deslizante que abarca *N* muestras por cada posición n, asumiendo que *N* es impar.

Las muestras se pueden indexar y escribir como un vector x(n) = [x_1(n),x_2(n),⋯,x_N(n)].
La muestra central de la ventana de observación se indica como $x_{(N+1)/2}$(n) y la estimación del filtro en dicha posición y(n).

El rango de muestras ordenadas se escriben como

$$x_{(1)}(n) \le x_{(2)}(n) \le \cdots \le x_{(N)}(n) \qquad \textbf{(3.1)}$$

3.1. Filtro CS

El filtro de realce que se presenta en esta sección permite realzar los contornos al mismo tiempo que suprime componentes tanto de ruido impulsivo y como algunas no impulsivas de ruido blanco aditivo. En añadido, éste se caracteriza por la sencillez de su implementación y la posibilidad de su uso en circunstancias de falta de conocimiento sobre el grado de borrosidad de la imagen, punto que le da un mayor atractivo al uso de este tipo de filtro.

La capacidad del filtro CS de eliminar gran parte de las componentes de ruido de una imagen entrante le confieren en una opción ideal para el realce de imágenes captadas mediante sónar. Dichas imágenes suelen contener gran cantidad de ruido, produciendo una distorsión en los contornos de las formas y creando un efecto de imagen borrosa o poco definida.

3.1.1. Definición

El filtro CS (Comparison and Selection) es un filtro espacial no lineal de realce que se caracteriza por eliminar las componentes impulsivas y realzar contornos a pesar de tender a distorsionar o eliminar pequeñas características de la señal. Proporciona una imagen final realzada cogiendo muestras alejadas de la media como valor resultante.

El filtro utiliza un tamaño de ventana impar descrito como 2*N*+1 para un entero positivo *N*, lo que permite evaluar el valor central de cada ventana.

La salida del CS se define como

$$Y_k = \begin{cases} X_k^{(N+1-J)}, & \text{si } \mu_k \geq M_k \\ X_k^{(N+1+J)}, & \text{en otro caso} \end{cases} \quad \textbf{(3.2)}$$

donde $X_k^{(i)}$ es la *i*-ésima muestra de menor valor dentro de la ventana de convolución, μ_k y M_k son la media estimada y el valor central de la ventana, respectivamente, y *J* es un entero tal que $1 \leq J \leq N$.

Cuando $J = N$, el filtro CS selecciona el mínimo ó máximo valor de la ventana dependiendo de los datos.

3.1.2. Propiedades

En este apartado se enumeran las propiedades que caracterizan al filtro CS, y se referencia a los artículos donde encontrar las comprobaciones pertinentes.
Se hace una separación entre propiedades deterministas y propiedades centradas en el realce de la imagen.

3.1.2.1. Propiedades deterministas

Las siguientes propiedades se pueden encontrar analizadas y probadas en [4] y [5].

Como se ha nombrado anteriormente, los filtros CS son no lineales, por lo que la propiedad de superposición no se cumple en su forma general, aunque sí bajo ciertas condiciones para combinaciones lineales de una señal arbitraria y una constante.

Se define a una secuencia de salida $\{Y_i\}$ de un filtro *S* como $\{Y_i\} = S(\{X_i\})$ para una secuencia de entrada $\{X_i\}$.

Propiedad 1 (Escala y tendencia a la invariancia): Para cualquier filtro CS,

$$S(a\{X_i\}+b\{1\}) = aS(\{X_i\}+b\{1\}) \quad \textbf{(3.3)}$$

si $\mu_i \neq M_i$ para toda *i*, donde {1} es la secuencia de valores constantes 1, y *a* y *b* son constantes.

Propiedad 2 (Desplazamiento de secuencias cóncavas y convexas): Si la entrada $\{X_i\}$ es estrictamente creciente (decreciente) y convexa (cóncava), entonces

$$S(\{X_i\}) = \{X_{i-J}\} \quad \textbf{(3.4)}$$

De forma similar, si $\{X_i\}$ es estrictamente creciente (decreciente) y cóncava (convexa), entonces

$$S(\{X_i\}) = \{X_{i+J}\}. \quad \textbf{(3.5)}$$

Para las siguientes propiedades se define que una raíz de un filtro es una secuencia invariable al filtrado. Como ejemplo, se muestra una raíz del filtro CS en (**Fig. 3.1 (a)**).

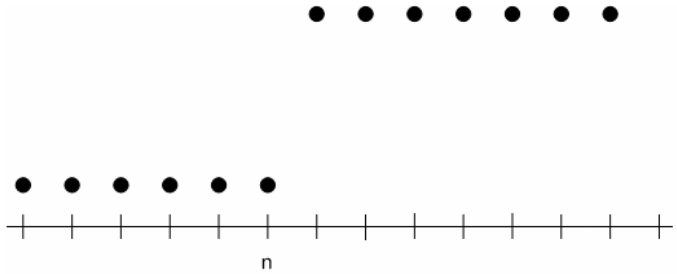

Fig. 3.1 (a) Ejemplo de un borde ideal, raíz del filtro CS

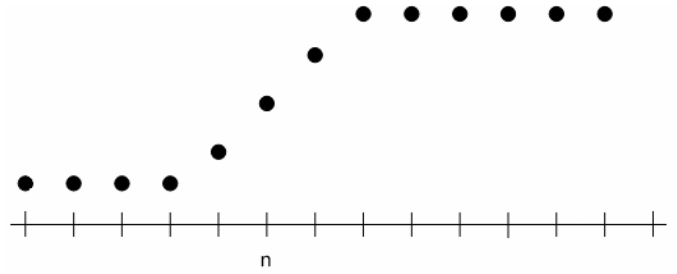

Fig. 3.1 (b) Ejemplo de un borde rampa

Propiedad 3 (Raíces por segmentos constantes): Una secuencia de segmentos constantes es una raíz del filtro CS con parámetro *J* si la duración de cada pulso, que son una sucesión de valores constantes, es mayor o igual que *N*+*J*.

Es necesario anotar que una secuencia que no sea de segmentos constantes puede ser una raíz de un filtro CS. La clase de secuencias de segmentos constantes con una mínima duración de pulsos *N*+*J* es un subconjunto del conjunto de raíces que incluye todas las raíces de los filtros CS.

Propiedad 4 (Raíces localmente monótonas): Si una secuencia monótona localmente de longitud $2N+1$ es una raíz del filtro CS, entonces ésta es una secuencia de segmentos constantes con duración de cada pulso mayor o igual a $2J$.

3.1.2.2. Propiedades de realce de contornos

Las propiedades que vienen a continuación se pueden encontrar analizadas y probadas en [4].

Se considera una secuencia $\{X_i\}$ que contiene un borde borroso, definido por

$$X_i = \begin{cases} 0, & i \le 0 \\ f_i, & 0 < i \le D \\ H, & D < i \end{cases} \qquad \textbf{(3.6)}$$

donde $\{f_i\}$, $-\infty < i < \infty$, es una secuencia que determina la forma del borde borroso. Las constantes positivas H y D indican el grado de borrosidad, siendo D un entero.

La secuencia $\{X_i\}$ representa un borde ideal cuando $D = 0$, y un borde rampa si $\{f_i\}$ tiene tendencia lineal con $f_0 = 0$ y $f_{D+1} = H$. En (**Fig. 3.1 (b)**) se ilustra un borde rampa.

Propiedad 5 (Realce de bordes rampa): Un borde rampa de duración D se convierte en un borde ideal después de $[E/J]$ y $[(D+1-N)/J]$ pasadas a través del filtro CS con $2N > D$ y $2N \le D$, respectivamente. Aquí $E = D/2$ si D es par, $E = (D+1)$ si D es impar, y $[x]$ es lo mismo que x si x es un entero y la parte entera de $x+1$, de lo contrario. La transición de 0 a H del resultante borde ocurre entre los tiempo índices E y $E+1$ donde $2N > D$, y entre $D+1-N$ y $D+2-N$ cuando $2N \le D$.

De esta propiedad se puede extraer que el filtro CS reduce la duración del borde de la imagen por cada filtrado en al menos J siempre que la duración sea mayor o igual que J, por lo que se realza el borde en cada pasada por el filtro.

Cuando se produce el borde ideal a partir de una rampa borde, es deseable obtener uno en el que la transición ocurra en medio de la rampa original. Esto ocurre cuando el filtro CS con $2N > D$ es usado, mientras este se desplaza a la derecha si $2N \le D$ porque $E < D+1-N$ para $2N \le D$.

El uso del filtro CS con una ventana mayor que la duración de la rampa es preferible en el realce de un borde rampas. Para un tamaño de ventana dado, el uso de un valor de J grande permite una convergencia más rápida hacia un borde ideal.

Propiedad 6 (Realce de bordes convexos/cóncavos): Cualquier borde convexo/cóncavo con duración *D* se convierte en un borde ideal después de al menos [*D*/*J*] pasadas a través del filtro CS con parámetro *J*.

Propiedad 7 (Realce de un borde con forma arbitraria): Después de al menos dos usos del filtro CS con $N \geq D$ y $D \leq J \leq N$, cualquier borde borroso se convierte en un borde ideal mientras $0 < f_i < H$ para $1 \leq i \leq D$.

Si el tamaño de la ventana de un filtro CS es considerablemente mayor que la duración del borde borroso, es posible seleccionar el parámetro J del filtro CS de forma que la salida siempre sea 0 ó H. Esto sucede si la ventana del filtro incluye en este caso un número suficiente de ceros y/o H en su interior.

Para la siguiente propiedad se considera una secuencia $\{X_i\}$ que representa un borde borroso definido por

$$X_i = \begin{cases} a_i, & i \leq 0 \\ g_i, & 0 < i \leq D \\ b_i, & D < i \end{cases} \qquad \textbf{(3.7)}$$

donde $\{a_i\}$, $\{g_i\}$, y $\{b_i\}$, $-\infty < i < \infty$, son secuencias tal que

$$\begin{gathered} -\varepsilon/2 \leq a_i \leq \varepsilon/2, \\ \delta + \varepsilon/2 \leq g_i \leq \delta + \varepsilon/2 + H, \text{ y} \\ 2\delta + \varepsilon/2 + H \leq b_i \leq 2\delta + 3\varepsilon/2 + H. \end{gathered} \qquad \textbf{(3.8)}$$

Aquí ε corresponde al parámetro de cantidad de ruido, y δ, H, y D son variables que miden el grado de borrosidad, y donde ε, δ, y H son reales positivos constantes y D es un entero positivo.

Propiedad 8 (Realce de bordes borrosos con ruido): Si $\delta > N_\varepsilon - H / 2$ y los parámetros N y J del filtro CS satisfacen $N \geq 2D$ y $D \leq J \leq N - D$, entonces después de dos o más usos del filtro CS, la salida $\{Y_i\}$ se representa como

$$Y_i = \begin{cases} a_i', & i \leq m \\ c_i', & i > m \end{cases} \qquad \textbf{(3.9)}$$

donde m es un entero, $1 \leq m \leq D$, y $\{a_i'\}$ y $\{c_i'\}$, $-\infty < i < \infty$, son secuencias cuyos valores están limitados a las mismas regiones que $\{a_i'\}$ y $\{c_i'\}$, respectivamente.

3.1.3. Características del filtro

El filtro CS posee características que lo convierten en una herramienta muy valiosa a la hora de mejorar la definición de los contornos de una imagen.
Uno de sus rasgos más importantes es el que hace referencia a la propiedad 1 vista anteriormente, se trata de la insensibilidad del filtro CS al ruido superpuesto a la señal de entrada. Esto lo hace posible su capacidad para eliminar componentes de ruido impulsivo y algunos de tipo no impulsivo blanco aditivo, lo cual se produce simultáneamente al realce del gradiente de los bordes. Este hecho es debido a que los picos de ruido suelen aparecer al final del conjunto de valores ordenados pertenecientes a la ventana del filtro, ya que se trata de valores extremos. En la (**Fig. 3.1**) se muestra una ventana del filtro CS.

Por tanto, el filtro CS puede suprimir al menos *N-J* impulsos en cualquiera de los dos lados de la muestra central de cada ventana. Si los impulsos no aparecen simultáneamente en ambos finales de la secuencia ordenada pero aparecen en sólo uno de los finales, los filtros CS pueden eliminar hasta *N* impulsos a pesar del parámetro *J*.

Esta situación se produce habitualmente en imágenes captadas por sónar, ya que es habitual la aparición de ruido resultado del proceso de medición, tanto por las condiciones ambientales como por los errores intrínsecos al equipo de medición.

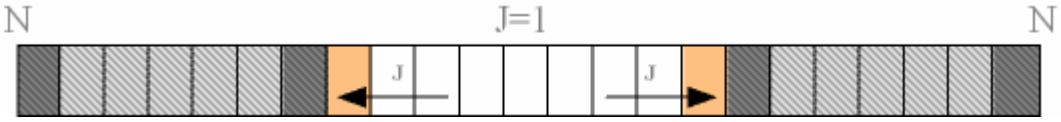

Fig. 3.1 Ventana ordenada del filtro CS con un valor de J=5. Los dos posibles valores salientes del filtro están resaltados en naranja.

El filtro CS combina características de supresión de ruido con las de realce de contornos incluyendo gran cantidad de tipos y condiciones, algunos de ellos tratados en el apartado de propiedades.

3.1.3.1. Elección de parámetros

El realce mediante el filtrado CS se puede obtener en diferente grado dependiendo de la cantidad de veces que hagamos uso de él, o del valor de los parámetros que se escojan al realizar el proceso. Es importante una correcta elección del tamaño de ventana y el parámetro *J*, ya que de ello depende que el resultado sea el óptimo.

La elección del tamaño de la ventana determina el realce a obtener a la salida del filtro, donde a mayor tamaño, mayor será la definición de los contornos en la imagen resultante, aunque por otro lado, mayor será la pérdida de los

pequeños detalles de la imagen entrante. La pérdida de detalles es debida a que al tratarse un mayor conjunto de muestras, los valores que caracterizan pequeñas zonas se ven eliminados por otros, provocando un suavizado en la imagen. Éste parámetro también influye en la rapidez de filtrado, ya que a mayor tamaño de ventana, mayor será el tiempo de ejecución, pudiendo hacer del filtrado un proceso lento. A cambio, posiblemente se obtenga un realce similar al que se obtendría con un mayor número de pasadas por un filtro CS con un valor de parámetro menor.

Otra característica a tener en cuenta es el acentuamiento del realce de una imagen que se produce al elegir un valor alto del parámetro *J*, ya que dentro de la secuencia ordenada de valores el valor resultante del filtrado será más extremo, es decir, más alejado de la media. Esta situación provoca una mayor definición de los bordes, a pesar de realzar también zonas donde hubiera componentes de ruido blanco aditivo que no pudieran ser eliminadas por el filtro.

Por lo tanto, la elección de los parámetros también determina la cantidad de veces que se filtrará una imagen entrante para llegar al resultado deseado.

De la propiedad 4 del apartado 3.1.2.1., se observa que para conseguir unas buenas características de realce de contornos se debe hacer uso repetidamente del filtro CS hasta obtener una raíz de dicho filtro. Sin embargo, uno de los problemas que pueden surgir es que la raíz resultante pueda tener bordes que no sean ideales, como sería el caso de un contorno formado por una serie de segmentos constantes. Además existe la posibilidad de que secuencias arbitrarias converjan en una raíz por el uso repetido del filtro CS, sobretodo si el parámetro *J* es un valor elevado.

En conclusión, se puede decir que el tamaño de la ventana de un filtro CS está limitado por las distancias entre bordes vecinos de una misma secuencia.
Como ya se dijo anteriormente, al seleccionar el parámetro *J* se necesita tener en consideración las ventajas y desventajas entre la elección de un mayor realce del gradiente del borde y una mayor supresión del ruido. Si es posible realizar el filtrado CS en más de una ocasión, es preferible elegir un valor de *J* razonablemente pequeño y repetir la operación, permitiendo eliminar en buena parte el ruido mientras se consigue un realce adecuado de los bordes. Sin embargo, no hay que olvidar que el filtro CS tiende a convertir todo segmento monótono de la señal de entrada en un borde ideal, por lo que hay que tener cuidado con un uso repetido del mismo.

Un aspecto a tener en consideración es la viabilidad de la implementación de un filtro, siendo de vital importancia que el algoritmo sea sencillo, tanto para facilitar el entendimiento del mismo, como para que su ejecución sea lo menos costosa posible. El filtro CS reúne esta condición, siendo su código simple e intuitivo, el cual se muestra en el Anexo C.

3.1.3.2. Módulo del filtro CS

El aspecto del módulo del filtro CS se muestra en (**Fig. 3.2**). La interfaz gráfica de usuario implementada contiene los dos parámetros correspondientes al filtro, los cuales son el tamaño de ventana y el parámetro A.

El tamaño de ventana ofrece un rango de números enteros de 3 (3x3) hasta 15 (15x15), incluyendo los números impares intermedios y ambos extremos.
El parámetro A, que corresponde al valor J, ofrece un rango de números enteros de 1 hasta 2*N*+3 / 2, ambos inclusive.
Los valores predeterminados de los parámetros son:

- *Tamaño de ventana (kernel size):* 3x3
- *Parámetro A:* 2

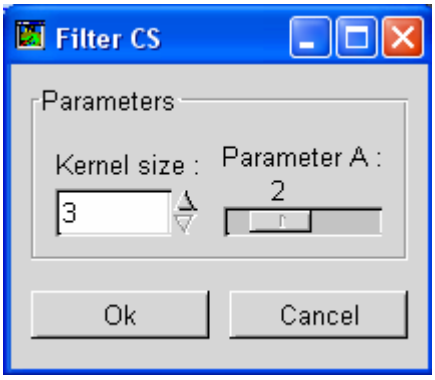

Fig. 3.2 Interfaz gráfica de usuario del filtro CS.

3.2. Filtro WMMR-MED

El filtro WMMR-MED tiene la capacidad de reestablecer un amplio abanico de bordes distorsionados a bordes ideales, suavizar regiones ruidosas sin bordes y eliminar tanto las componentes impulsivas del ruido como las componentes no impulsivas de ruido aditivo.

3.2.1. Definición

Los filtros WMMR (Weighted Majority of m values with Minimum Range) son unos filtros espaciales no lineales de realce que tienen la característica de eliminar las componentes impulsivas, preservar los contornos perfectos y realzar los imperfectos.

Para obtener el resultado se debe encontrar las (*N*+1)/2 muestras más próximas entre ellas, es decir *i* debe ser encontrada tal que $x_{((N-1)/2+i)} - x_{(i)}$ sea un mínimo.

La salida del filtro WMMR es dado por

$$y = F_{WMMR}(x) = \sum_{j=i}^{(N-1)/2+i} w_j x_{(j)}. \qquad \textbf{(3.10)}$$

Para el realce de los contornos la elección de pesos escogida es

$$w_j = \begin{cases} 1, & \text{si } j = (N-1)/4+i \\ 0, & \text{en otro caso.} \end{cases} \qquad \textbf{(3.11)}$$

El filtro proporciona como resultado la media de las $(N+1)/2$ muestras más cercanas de la ventana, por lo que es llamado WMMR-MED.

Por lo tanto, el procedimiento seguido es la búsqueda de un grupo de muestras de la ventana del filtro que tengan valores muy cercanos, obtener su media y asignársela a la muestra central de la ventana una vez ha sido multiplicada por un factor de escala, variando éste del 0 al 1.

El resultado obtenido por el filtro WMMR-MED viene determinado por

$$y = (1-f_x)* m_x + f_x * x_i \qquad \textbf{(3.10)}$$

donde f_x es el factor de escala introducido, $0 \leq f_x \leq 1$, m_x es la media del conjunto de $N+1$ valores de la ventana con menor rango y x_i es el valor de la muestra entrante.

3.2.2. Propiedades

A continuación se presentan las propiedades del filtro WMMR-MED [9].

Sea x un vector y J un entero, $1 \leq J \leq N+1$, tal que $[x_{(J+N)} - x_{(J)}] \leq [x_{(j+N)} - x_{(j)}]$ para $1 \leq j \leq N+1$, asumiendo que J es único. Esta suposición se debe evitar en el procesado de señales digitales ya que al tratar con valores dentro de un rango finito hay posibilidades de que J no sea único. Por lo tanto se define lo siguiente:

***Definición*:** Para un segmento x_i de longitud $2N+1$ de una secuencia $\{x_i\}$ se define $x^j = x_{(N+j)} - x_{(j)}$, $1 \leq j \leq N+1$. Sea $x = min\{x^j, 1 \leq j \leq N+1\}$, entonces se define

$$Px_i = \{A : A \text{ es un conjunto de } N+1 \text{ valores de } x_i \text{ con rango } x\}. \quad \textbf{(3.11)}$$

Para el vector $w = (w_1, w_2, \cdots, w_{N+1})$ se define WMMR (x_i) = med{$w*a$: el producto escalar de w y a, donde a es el vector de los elementos ordenados algebraicamente de A, y A un elemento de Px_i}. En adelante será reducido a WMMR (x_i) = med{$w*a$: a tiene el mínimo rango}.

A continuación se enumeran un conjunto de propiedades que caracterizan a la familia de filtros WMMR .

Propiedad 1: Los filtros WMMR y mediana producen un idéntico resultado para cualquier señal entrante con dos posibles valores si la suma de sus pesos es una. En adelante se hará referencia a este filtro, llamado WMMR normalizado.

En caso de señales de longitud finita, dos únicos valores posibles y puntos que la limitan, los puntos fijos del WMMR normalizado son un segmento constante de longitud mínima N+1.

Propiedad 2**:** Los segmentos constantes mantienen la misma amplitud y no decrecen en longitud al ser pasados por el filtro normalizado WMMR.

Propiedad 3**:** Asumiendo pesos no negativos y que $\{x_1, x_2, \cdots, x_{2N+2}\}$ es tal que $x_{2N+2} = \max\{x_1, x_2, \cdots, x_{2N+2}\}$ y $x_1 = \min\{x_1, x_2, \cdots, x_{2N+2}\}$, entonces WMMR$(x_1) \leq$ WMMR(x_2).

Propiedad 4**:** Los bordes perfectos son puntos fijos para el filtrado normalizado WMMR con pesos no negativos y $w_1 = w_{N+1}$.

Propiedad 5: Asumiendo una señal con un segmento constante de longitud mínima N+1, cada segmento de longitud 2N+1 tiene al menos una de las siguientes características:

- Contiene un segmento constante de longitud N+1.
- El segmento de 2N+3, siendo este el original incrementado por un punto en cada dirección, es no creciente o no decreciente.

Propiedad 6**:** La salida del filtrado WMMR no depende del orden temporal que tengan los valores de la ventana.

Propiedad 7: Sea x = WMMR (x_i) con vector peso $w = (w_1, w_2, \cdots, w_{N+1})$ y w_i no negativo para cada i. Si $j = \min\{1, 2, \cdots, N+1\}$ tal que $w_j > 0$, y $x = \max\{x_i, x_{i+1}, \cdots, x_{i+2N}\}$, en tal caso existen al menos (N+1)-j+1 valores en x_i con valor x.

La siguiente propiedad es aplicable a todos los filtros WMMR normalizados y simétricos, incluyendo en consecuencia al filtro WMMR-MED.

Definición**:** Se dice que los N+1 pesos $w = (w_1, w_2, \cdots, w_{N+1})$ son simétricos si $w_1 = w_{N+2-i}$ para $i = 1, 2, \cdots, N+1$.

***Propiedad 8*:** Para señales con valores múltiples se obtiene el siguiente resultado sobre el filtrado WMMR si los pesos son normalizados.

(a) Los impulsos son eliminados tras una pasada por el filtro.

Si los pesos son normalizados y no negativos:

(b) Si la entrada es una señal localmente monótona de grado M, para una M mayor o igual que N+2, la salida será una señal localmente monótona de grado M.

Si los pesos son normalizados, no negativos y simétricos, entonces:

(c) Los bordes no perfectos se aproximan asintóticamente al borde perfecto por cada aplicación iterativa del filtro.

Las propiedades anteriores, aplicables a todos los filtros WMMR con valores normalizados y simétricos, hacen que estos eliminen las componentes de ruido impulsivas y conserven los bordes perfectos de igual forma que el filtro mediana.

La siguiente propiedad diferencia a los filtros WMMR de las características del filtro mediana, ya que le permite realzar contornos imperfectos de forma que la salida del filtrado converja hacia el borde perfecto tras aplicarlo de forma repetida.

Se asume que N es par y se considera el filtro WMMR-MED, el cual reemplaza el valor central de la ventana por la media del conjunto de N+1 valores de la ventana con menor rango. Se considera de igual forma un borde $(c_1)^{N+1}$, x_i, $(c_2)^{N+1}$, donde la mitad de la secuencia x_i se encuentra entre $(c_{1+}\ c_2)/2$ y c_1, y la otra mitad entre $(c_{1+}\ c_2)/2$ y c_2, denominando como E a un borde de estas características.

***Propiedad 9*:** Una pasada por el filtro WMMR-MED transforma un borde imperfecto de la forma E en un borde ideal.

3.2.3. Características

El filtro WMMR-MED es adecuado para el procesado de imágenes con una cantidad de ruido importante debido a sus características para eliminarlo y una detección de bordes suave. Esto se debe a que este filtro posee las mismas propiedades que el filtro mediana además de la capacidad de realzar bordes que han sido distorsionados por el ruido de la imagen, transformándolos en bordes ideales.

Por lo tanto, este filtro es capaz de eliminar tanto componentes de ruido impulsivo, como de tipo no impulsivo blanco aditivo, hecho que se produce simultáneamente al realce del gradiente de los bordes.

El proceso para la obtención de una imagen resultante donde los bordes imperfectos y distorsionados lleguen a ser bordes ideales puede durar solamente un filtrado, a causa de la última propiedad vista en el apartado anterior, donde este filtro es capaz de transformar un tipo de borde imperfecto en un borde marcado y definido filtrándola una sola vez.

Cada pasada que reciba la imagen resultante por el filtro WMMR-MED producirá un mayor realce y una mejora en su definición, permitiendo por lo tanto elegir los parámetros en función de las veces que es posible filtrar cierta imagen.

3.2.3.1. Elección de parámetros

El filtro WMMR-MED es capaz de transformar bordes ruidosos e imperfectos en ideales en un único filtrado dependiendo del tipo de bordes y de la elección de los parámetros.

Se disponen de dos parámetros diferentes, la combinación de los cuales posibilita el filtrar la imagen con unos objetivos determinados, ya sea un mayor o menor grado de definición, como las pasadas por el filtro que se desean ejecutar o la velocidad de cada filtrado.

El primero de ellos es el tamaño de la ventana del filtro, cuya elección de valor determina el realce de los bordes en la imagen resultante, así como una mayor o menor pérdida de los pequeños detalles. Cuanto mayor es el valor del parámetro, mayor es la definición de los bordes, y de igual forma se aumenta el nivel de pérdida de detalles, circunstancia que se debe al mayor número de muestras a tratar en el filtrado por cada ventana. De éste valor también depende el número de pasadas por el filtro que debe realizar la imagen y el tiempo de ejecución de cada filtrado, ya que a mayor cantidad de muestras a tratar por cada ventana, mayor es el tiempo necesario para el proceso de filtrado y menor el número de veces a filtrar para obtener un resultado similar.

El factor de escala corresponde al otro parámetro del filtro, el cual determina el grado de realce y definición a alcanzar en cada filtrado WMMR-MED. Su valor puede oscilar entre 0 y 1, suprimiendo más ruido y definiendo mejor los bordes cuanto menor es el parámetro, todo ello a coste de una pérdida de definición. Esto permite que al seleccionar un valor pequeño de factor de escala sea posible realzar los bordes de forma efectiva con un simple filtrado.

El filtro WMMR-MED ofrece un amplio rango de grados de definición de los bordes, eliminación y suavizado de zonas ruidosas, y realce de bordes imperfectos con una elección correcta de sus parámetros, en especial el factor de escala que es capaz de ofrecer una imagen resultante con un alto grado de realce si su valor es próximo a 0.

El código del filtro WMMR-MED se puede encontrar implementado en el Anexo C.

3.2.3.2. Módulo del filtro WMMR-MED

El aspecto del módulo del filtro WMMR-MED se muestra en (**Fig. 3.3**). La interfaz gráfica de usuario implementada contiene los dos parámetros correspondientes al filtro, los cuales son el tamaño de ventana y el parámetro A.

El tamaño de ventana ofrece un rango de números enteros de 3 (3x3) hasta 15 (15x15), incluyendo los números impares intermedios y ambos extremos.
El parámetro A, que corresponde al factor de escala, ofrece un rango de números con dos decimales de 0 hasta 1, ambos inclusive.
Los valores predeterminados de los parámetros son:

- *Tamaño de ventana (kernel size):* 3x3
- *Parámetro A:* 0.2

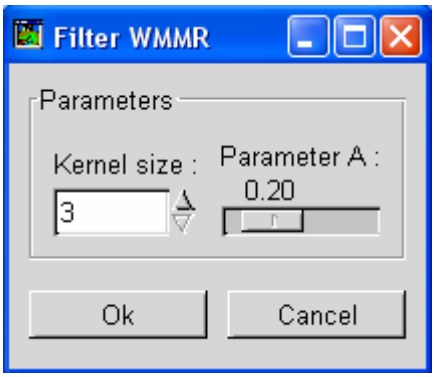

Fig. 3.3 Interfaz gráfica de usuario del filtro WMMR-MED.

3.3. Filtro Volterra

El filtro analizado en esta sección posee la característica de realzar la imagen dependiendo de la media de intensidad que tienen los píxeles locales, consiguiendo un realce más exhaustivo en zonas donde los detalles son menos perceptibles al ojo humano y reduciéndolo en las zonas donde el realce del ruido podría afectar en gran medida a la calidad de la imagen.

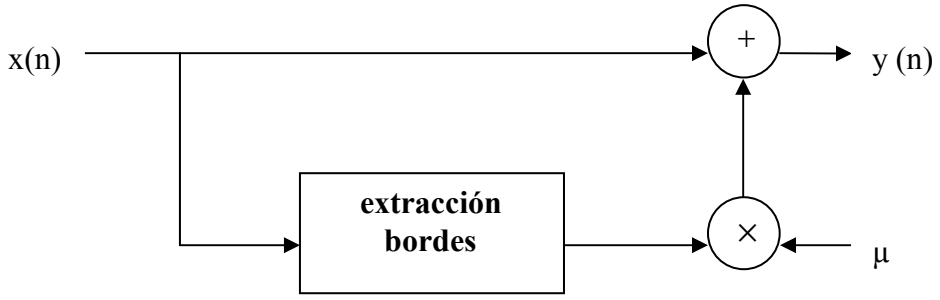

Fig. 3.4 Diagrama bloque del filtro Volterra

El esquema usa un filtro cuadrático Volterra que extrae la información de los bordes de la imagen, para después multiplicados por un parámetro añadirla a la imagen original, como se muestra en (**Fig. 3.4**), donde se muestra el esquema del filtro implementado.

El realce obtenido por el filtro Volterra es equivalente al producto de la media estimada local y un filtro paso alto. Además éste es capaz de discernir entre información de la imagen y el ruido, evitando realzar detalles formados a consecuencia de la aparición de ruido, como sí haría un filtro lineal.

Otro punto no menos importante es el menor gasto computacional que requiere un filtro Volterra respecto a un filtro lineal con características de realce parecidas, aspecto que lo hace indicado para su implementación y uso en el realce de imágenes captadas por sónar, donde el cálculo computacional a realizar es considerable.

3.3.1. Definición

Un filtro cuadrático Volterra discreto en el tiempo es definido por (ver [6] y [12])

$$y(n) = V_2(x) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} h_2(k_1,k_2)x(n-k_1)x(n-k_2)\,. \qquad \textbf{(3.12)}$$

La expresión se puede interpretar como una convolución bidimensional de $h_2(n_1,n_2)$ con productos de las muestras de la secuencia entrante.

Éste hecho se observa de forma más clara si se define

$$\overline{x}(n_1,n_2) = x(n_1)x(n_2)\,. \qquad \textbf{(3.13)}$$

Entonces se puede escribir

$$y(n) = h_2(n_1, n_2) ** \overline{x}(n_1, n_2)\big|_{n=n_1=n_2} \quad \textbf{(3.14)}$$

donde $**$ indica una convolución bidimensional.

Se define el espectro de los kernels Volterra como una transformada multidimensional de Fourier de $h_2(n_1, n_2)$

$$H_2(\omega_1, \omega_2) = F\{h_2(n_1, n_2)\} = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} h_2(n_1, n_2) e^{-j(n_1\omega_1 + n_2\omega_2)} \quad \textbf{(3.15)}$$

El símbolo F indica la transformada de Fourier en una o más dimensiones, dependiendo del contexto. Se denomina $H_2(\omega_1, \omega_2)$ a la respuesta frecuencial generalizada de segundo orden, así como a $h_2(n_1, n_2)$ la respuesta impulsiva de segundo orden.

Se puede reescribir **(3.14)** usando las respuestas frecuenciales de la entrada y el sistema como

$$y(n) = F^{-1}\{H_2(\omega_1, \omega_2)\overline{X}(\omega_1, \omega_2)\}\big|_{n=n_1=n_2} \quad \textbf{(3.16)}$$

donde $X(\omega) = F\{x(n)\}$ y $\overline{X}(\omega_1, \omega_2) = F\{\overline{x}(n_1, n_2)\} = X(\omega_1)X(\omega_2)$. $X(\omega) = F\{x(n)\}$.

Al ser un filtro no lineal, la respuesta impulsiva de los filtros cuadráticos Volterra no define necesariamente el filtro de forma unívoca, ya que pueden existir diversos kernels con el mismo filtro. Sólo si el kernel es simétrico define de forma unívoca el sistema, y para un sistema de segundo orden, esto significa que

$$h_2^s(n_1, n_2) = h_2^s(n_2, n_1)\,. \quad \textbf{(3.17)}$$

En el caso de las imágenes, donde las señales de entrada y salida son secuencias bidimensionales, la expresión del filtro cuadrático Volterra discreto en el tiempo es

$$y(n_1, n_2) = \sum_{k_1} \sum_{k_2} \sum_{k_3} \sum_{k_4} h_2(k_1, k_2, k_3, k_4) \times x(n_1 - k_1, n_2 - k_2) x(n_1 - k_3, n_2 - k_4)$$

$$= h_2(n_1, n_2, n_3, n_4) *_4 \overline{x}(n_1, n_2, n_3, n_4)\Big|_{\substack{n_1=n_3 \\ n_2=n_4}} \qquad \textbf{(3.18)}$$

con $\overline{x}(n_1, n_2, n_3, n_4) = x(n_1, n_2)x(n_3, n_4)$, y donde se han omitido los límites de las variables $k_1, \cdots, k_4$, siendo en el caso más general desde – ∞ hasta ∞.

En [6], se analiza una subclase de filtros cuadráticos Volterra de la que se deriva una versión 2-D del filtro Teager [7], la cual da una señal de salida constante $y(n_1, n_2) = \sin^2(\omega_0)$ para una entrada sinusoidal independientemente de la orientación de la excitación que reciba. El filtro resultante obtenido es una aproximación al filtro Teager ideal, ya que muestra un comportamiento similar además de resultar considerablemente más sencillo de implementar. La ecuación del filtro cuadrático Volterra obtenido es

$$\begin{aligned} y(n_1, n_2) = {} & 3x^2(n_1, n_2) - \frac{1}{2}x(n_1+1, n_2+1) \times x(n_1-1, n_2-1) \\ & -\frac{1}{2}x(n_1+1, n_2-1)x(n_1-1, n_2+1) \\ & -x(n_1+1, n_2)x(n_1-1, n_2) - x(n_1, n_2+1)x(n_1, n_2-1) \end{aligned} \qquad \textbf{(3.19)}$$

siendo ésta la expresión usada en el algoritmo del filtro implementado.

3.3.2. Propiedades

A continuación se define una subclase de filtros cuadráticos Volterra y se muestran sus propiedades [6], de la cual deriva el filtro implementado en esta sección. Los filtros de dicha clase se pueden aproximar como un producto de la media local estimada y un filtro paso alto.

Se modela una secuencia de entrada *x*(n) mediante un proceso con distribución uniforme e idénticamente independiente, una media μ_x y una desviación estándar σ_x.

Teorema: un filtro Volterra unidimensional de segundo orden se puede aproximar por

$$Y(\omega) \approx 2\mu_x X(\omega) H_2(\omega, 0) \qquad \textbf{(3.20)}$$

si

$\mu_x > \sigma_x$

$$H_2(0,0) = \sum_{k_1}\sum_{k_2} h_2(k_1,k_2) = 0$$

$$H_2(\omega,0) = H_2(0,\omega)$$

$$S^h = \sum_{\substack{k_1 \\ k_1 \neq k_2}}\sum_{k_2} h_2(k_1,k_2)[2h_2(k_1,k_1) + h_2(k_2,k_2)] + \sum_{\substack{k_1 \\ k_1 \neq k_2 \\ k_2 \neq k_3}}\sum_{k_2}\sum_{k_3} h_2(k_1,k_2)\times[h_2(k_1,k_3) + h_2(k_2,k_3)] \geq 0 \text{.} \qquad \textbf{(3.21)}$$

El error aproximado es dado por

$$Y_e(\omega) = \frac{1}{2\pi}\int_0^{2\pi} H_2(\omega_1,\omega-\omega_1)\hat{X}(\omega_1)\hat{X}(\omega-\omega_1)d\omega_1 \qquad \textbf{(3.22)}$$

donde $\hat{X}(\omega) = X(\omega) - 2\pi\mu_x\delta(\omega)$ es la versión de $X(\omega)$ con la media eliminada.

La expresión **(3.21)** se puede simplificar en ciertos casos. En la siguiente clase los filtros consisten en cuadrados de píxeles.

Definición: Si $h_2(n_1,n_2) = 0$ para todo $n_1 \neq n_2$, $\sum_{n_1}\sum_{n_2} h_2(n_1,n_2) = 0$ y $H_2(\omega,0)$ tiene características paso alto, entonces se denomina filtro cuadrático Volterra discreto en el tiempo de clase I.

Lema1: Los filtros clase I se pueden aproximar al producto de la media estimada local y un filtro paso alto.

Definición: Si $h_2(n_1,n_2) = 0$ para todo $n_1 \neq -n_2$, $\sum_{n_1}\sum_{n_2} h_2(n_1,n_2) = 0$ y $H_2(\omega,0)$ tiene características paso alto, entonces el filtro se denomina filtro cuadrático Volterra discreto en el tiempo de clase II.

Lema2: Un filtro de clase II se puede aproximar al producto de la media estimada local y un filtro paso alto

2.3.2.1. Propiedades filtro bidimensional

En [6], se analizan las propiedades del filtro cuadrático Volterra para señales bidimensionales que se muestran a continuación, las cuales son una extensión de las propiedades para señales unidimensionales.

El filtro Volterra finalmente implementado corresponde a la clase II de filtros Volterra, la cual tiene como propiedad la simetría de sus sistemas [6], preservando de esta forma la localización de los bordes cuando se aplica a

imágenes. Si éste no fuera el caso, no se podrían realzar los bordes ya que no se conocería la localización exacta del detalle.

El bloque fundamental de construcción de sistemas de clase II es $x(n-k)x(n+k)$. En conjunto el sistema consiste en una combinación lineal de diferentes de estos términos para diferentes *k*. Se extiende esta idea a un caso bidimensional simplemente definiendo el bloque de construcción básico de un filtro bidimensional como $x(n_1-k_{1,}n_2-k_2)x(n_1+k_{1,}n_2+k_2)$, el cual consiste en un conjunto de píxeles centrados alrededor del píxel actual $x(n_1,n_2)$.

Usando la propiedad de los filtros Volterra que enuncia la linealidad de sus kernels y considerando el bloque de construcción básico anteriormente visto, se presenta el concepto de filtros básicos para el diseño de filtros Volterra.

Se define la siguiente expresión como base para los filtros de clase II

$$y_{k_1,k_2}(n_1,n_2)=x^2(n_1,n_2)-x(n_1+k_1,n_2+k_2)\times x(n_1-k_1,n_2-k_2) \qquad \textbf{(3.23)}$$

donde $0\le k_1<\infty$, $-\infty<k_2<\infty$, y $k_1+|k_2|\neq 0$, por ejemplo, k_1 y k_2 no pueden ser iguales a 0 simultáneamente. Se consideran parejas únicas de k_1 y k_2.

Extendiendo las propiedades para señales unidimensionales, para la respuesta impulsiva se obtiene la propiedad

$$h_2(n_1,n_2,n_3,n_4)\neq 0 \text{ sólo si } n_1=-n_3 \text{ y } n_2=-n_4. \qquad \textbf{(3.24)}$$

Análogamente al Lemma2 [6], también se requiere que

$$\sum_{k_1}\sum_{k_2}\sum_{k_3}\sum_{k_4}h_2(k_1,k_2,k_3,k_4)=H_2(0,0,0,0)=0. \qquad \textbf{(3.25)}$$

Además también se puede mostrar que

$$H_2(\omega_1,\omega_2,0,0)=H_2(0,0,\omega_1,\omega_2). \qquad \textbf{(3.26)}$$

La demostración se puede encontrar en [6].

Usando **(3.24)** y **(3.26)** se llega a la extensión de **(3.20)**, siendo la expresión

$$Y(\omega_1,\omega_2) \approx 2\mu_x X(\omega_1,\omega_2) H_2(\omega_1,\omega_2 0,0)\,. \qquad \textbf{(3.27)}$$

Asumiendo que $H_2(\omega_1,\omega_2 0,0)$ tiene características de paso alto en cualquiera de las direcciones ω_1 o ω_2, o ambas, se concluye que los sistemas de clase II bidimensionales se pueden aproximar como un producto de la media estimada local y un filtro paso alto.

3.3.3. Características

El filtro cuadrático de segundo grado Volterra es un filtro no lineal capaz de mejorar en muchos casos el resultado de un filtro lineal de forma considerable y con un menor coste computacional.

El realce obtenido con este filtro es similar al que se obtendría mediante el producto de la media estimada local y un filtro paso alto. Además, el filtro Volterra es capaz de discernir entre información de la imagen original y el ruido añadido, evitando el realce del segundo, y en consecuencia un deterioro de la imagen resultante.

El filtro Volterra realza la imagen dependiendo de la intensidad media de los píxeles locales, es decir, se basa en las propiedades descritas en la ley de Weber-Fechner (ver Anexo A). Éste método realza en mayor grado las regiones con más brillo, donde el ruido y otras fluctuaciones de los niveles de gris son mucho menos visibles que en las regiones oscuras, donde es preferible reducir o eliminar el realce debido a que podría deteriorar la calidad de la imagen. Por lo tanto, el filtro Volterra realza las diferentes zonas de la imagen en mayor o menor grado basándose en la capacidad de percepción del ojo humano, consiguiendo así una imagen de mayor calidad visual.

El ahorro en el coste computacional es de suma importancia a la hora de tratar con imágenes de gran tamaño, lo que supone un ahorro en el tiempo de procesado y en la potencia computacional requerida. En (**Fig. 3.4**) se presenta el esquema global del filtro, donde el bloque de detección de bordes comúnmente formado por un producto del filtro paso alto y el promedio local de los píxeles, es sustituido por el filtro Volterra. Con esta substitución se consiguen unos resultados similares con una considerable reducción del número cálculos.

3.3.3.1. Elección de parámetros

El filtro Volterra permite modificar el grado de realce mediante la elección de dos parámetros y sus respectivas combinaciones.

El tamaño de la ventana del filtro es el primer parámetro a tener en cuenta, y es común en todos los módulos. El valor de este parámetro determina el nivel de

realce a obtener, a la vez que el coste computacional que requerirá el filtrado. En este filtro la elección de dicho valor es de suma importancia, ya que debido a las características del filtro cuadrático Volterra, para ventanas de gran tamaño se hace necesario un gran número de operaciones computacionales. Cuanto mayor sea el valor, más realce y definición se obtiene en la imagen resultante.

El segundo de ellos es el factor de escala con el que se multiplica la salida del filtro Volterra, para posteriormente agregarlo al valor de la imagen original de entrada y producir la imagen final, lo cual se muestra en (**Fig. 3.4**). Éste parámetro está implementado como el Parámetro de Filtro A, el cual puede variar de 0.001 a 0.1, siendo el valor por defecto de 0.005. Al elegir un valor del Parámetro A elevado, se aumenta el valor del producto escalado a sumar al valor de la imagen original, por lo que el realce de los bordes aumenta. La elección de un valor del Parámetro A cerca del mínimo produce una imagen de notable nitidez, consiguiendo una imagen realista. Por el contrario, si se establece un valor mayor que 0.005 se producen imágenes con altos contrastes, enfatizando los bordes.

Un punto a destacar de este filtro es la solución encontrada a la difícil implementación de los filtros cuadráticos Volterra. El uso de ésta subclase del filtro Volterra permite obtener unos resultados equivalentes con un código más sencillo de implementar. El código del filtro Volterra se muestra en el Anexo C.

3.3.3.2. Módulo del filtro Volterra

El aspecto del módulo del filtro Volterra se muestra en (**Fig. 3.5**). La interfaz gráfica de usuario implementada contiene los dos parámetros correspondientes al filtro, los cuales son el tamaño de ventana y el parámetro A.

El tamaño de ventana ofrece un rango de números enteros de 3 (3x3) hasta 15 (15x15), incluyendo los números impares intermedios y ambos extremos.
El parámetro A, que corresponde al factor de escala, ofrece un rango de números con tres decimales de 0 hasta 0.1, ambos inclusive.
Los valores predeterminados de los parámetros son:

- *Tamaño de ventana (kernel size):* 3x3
- *Parámetro A:* 0.005

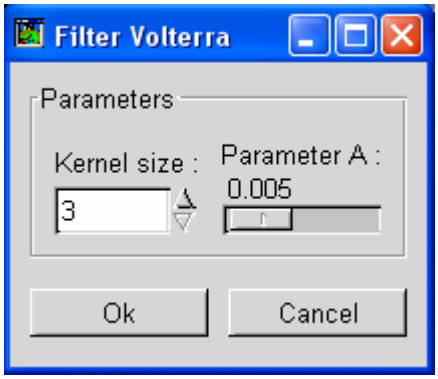

Fig. 3.5 Interfaz gráfica de usuario del filtro Volterra.

CAPÍTULO 4. Software

El objetivo final de este trabajo es implementar una serie de módulos, compuestos por filtros espaciales de realce, dedicados a complementar la aplicación OpenEV.

OpenEV es un programa con licencia de código abierto que forma parte de diferentes paquetes de software gratuitos, encaminados al manejo y procesado de rasters y vectores de datos.

La distribución elegida para este trabajo es el paquete de software FWTools.

4.1. FWTools

La distribución FWTools es la opción más recomendable para el uso del programa OpenEV, debido a que posee un completo conjunto de paquetes y está disponible tanto para Windows, como para Linux.

FWTools es un conjunto de GIS ("Geographic Information Systems") de código abierto que incluye recientes versiones de OpenEV, MapServer, GDAL, PROJ.4 y OGDI, así como diversos paquetes de soporte para estos. FWTools también incluye una completa distribución de Python, y un paquete de Python que da soporte a GDAL y MapServer.

La versión de la distribución utilizada durante el desarrollo de los módulos es FWTools 1.3.1, la cual es compatible con la más reciente aparecida hasta el momento, la 1.4.2.
La versión de Python utilizada en el paquete FWTools es la 2.3.4, siendo la misma tanto en la versión usada como en las posteriores.

4.1.1. Subpaquetes

La distribución FWTools se compone de una serie de paquetes que proporcionan diferentes funcionalidades:

- *OpenEV*: Una potente herramienta de análisis y visualización de rasters y/o vectores de datos.

- *MapServer*: Un paquete de mapeo web.

- *GDAL/OGR*: Una librería y un conjunto de aplicaciones con utilidades de líneas de comando para la lectura y escritura de varios formatos de rasters (GDAL) y vectores (OGR) geoespaciales.

- *PROJ.4*: Una librería de proyecciones cartográficas con utilidades de líneas de comando.

- *OGDI*: Una tecnología de lectura multiformato de rasters y vectores ideal para la inclusión de soporte para varios formatos militares incluyendo VPF (por ejemplo VMAP, VITD), RPF (por ejemplo CADRG, CIB) y ADRG.

- *Python*: Un lenguaje de scripting.

4.2. OpenEV

OpenEV es una aplicación formada por un conjunto de librerías software, creada para la visualización y análisis de vectores y/o rasters de datos geospaciales.

La elección de OpenEV como software donde implementar los módulos de filtros de realce se justifica por su libre distribución, su soporte multiplataforma y las funcionalidades que proporciona al usuario.

Se trata además, de una aplicación en constante mejora debido a su uso en compañías privadas, universidades, gobiernos y organizaciones sin ánimo de lucro de todo el mundo.

4.2.1. Características de OpenEV

El software se distribuye bajo la licencia de código abierto GNU LGPL, lo cual permite el acceso a un mayor número de usuarios y a su vez, posibilita que un mayor número de programadores se impliquen en el proyecto.

OpenEV tiene soporte multiplataforma, lo que flexibiliza y generaliza su empleo entre los usuarios. La aplicación está disponible para plataformas tan populares como Linux, Windows, Solaris o IRIX.

A continuación se enuncian las funciones relativas al procesado más destacadas de OpenEV:

- Soporta el procesado de vectores y rasters de datos geoespaciales de gran tamaño, del orden de gigabytes.

- Permite el procesado de canales múltiples y bases de datos de rasters complejas.

- Entiende e interpreta información georeferenciada y proporciona reproyecciones de bases de datos al momento.

- Proporciona funciones para manipular la visualización a tasas de imágenes interactivas

- Se trata de una potente herramienta de análisis de imágenes y visualización 2D y 3D.

4.2.2. Funcionamiento y utilidades de OpenEV

La herramienta OpenEV ofrece la posibilidad de adecuarla a las necesidades requeridas por el usuario al permitir la ocultación de las diferentes funcionalidades existentes, así como la añadidura de módulos propios con nuevas funcionalidades.

La personalización de la apariencia y su funcionalidad no requiere reescribir la aplicación principal o el código de la ventana de visualización, debido a que se pueden usar archivos de configuración XML para seleccionar cuáles de los iconos y menús disponibles de inicio serán mostrados en OpenEV, una vez la interfaz del usuario sea lanzada por la aplicación principal, o en la shell de Python. Por el contrario, los archivos de configuración XML no pueden ser usados para crear nuevas callbacks (retrollamadas), sino que sólo pueden modificar cómo las callbacks serán referenciadas.

La funcionalidad del programa se puede extender mediante el uso de la clase Tool_GViewApp. El caso más general de la clase sólo contiene una referencia a la instancia OpenEV (GViewApp) que se encuentra en uso, un conjunto de entradas a añadir al menú y barras de iconos de la interfaz de usuario, así como la shell de Python.
Todas las entradas de menú están definidas por un string por defecto que describe su ubicación, una posición entera relativa a los ítems existentes en el mismo menú y una callback (ver códigos en Anexo C). La referencia a la aplicación principal permite a la herramienta interactuar con las imágenes y capas en uso, además de editar la barra de herramientas. Es posible personalizar cualquier aspecto de OpenEV usando XML excepto las callbacks, ya que éstas deben ser localizadas a partir de la información existente en el archivo XML, ya que el código de la herramienta de las callback no forma parte de la clase gvviewwindow.

4.2.3. Herramientas (Plug-ins)

OpenEV permite añadir un enlace a las funciones callback propietarias del usuario a través de la aplicación principal usando la clase Tool_GViewApp situada en gviewapp.py. La clase por defecto simplemente almacena una referencia a la aplicación principal, objetos que almacenan la vista y el menú pyshell, e iconos de inicio añadidos por la herramienta.

4.2.4. Tecnologías aplicadas en OpenEV

OpenEV hace uso de diversas tecnologías, destacando las siguientes:

- *OpenGL:* Una especificación estándar usada para aumentar la velocidad del renderizado de vectores y rasters en entornos acelerados. El mapeo de texturas es usado para la visualización de los rasters.

- *Python:* Lenguaje de programación que proporciona diversas componentes y aplicaciones de OpenEV, a pesar de que el núcelo del programa está implementado en C.

- *Numeric Python:* Se trata de un objeto orientado, un lenguaje de escritura matemática de código abierto usado para crear una potente herramienta de análisis de imágenes dentro del entorno OpenEV.

- *GTK+ (GIMP Toolkit):* Es un conjunto importante de bibliotecas y rutinas usado para implementar la interfaz gráfica de usuario (GUI), y el componente del núcleo de OpenEV. GTK se distribuye bajo la licencia LGPL y proporciona portabilidad, además de componentes GUI sofisticados. Muchos de los desarrollos de niveles GUI están hechos en Python usando el Python GTK bindings.

- *GDAL:* La biblioteca GDAL es usada para obtener acceso en la ejecución de los diversos formatos de rasters geoespaciales.

- *PROJ.4:* La librería PROJ.4 es usada para dar soporte a las proyecciones subyacentes de imágenes.

En (**Fig. 4.1**) se muestra un esquema de las dependencias entre las diferentes tecnologías usadas en OpenEV, así como su organización.

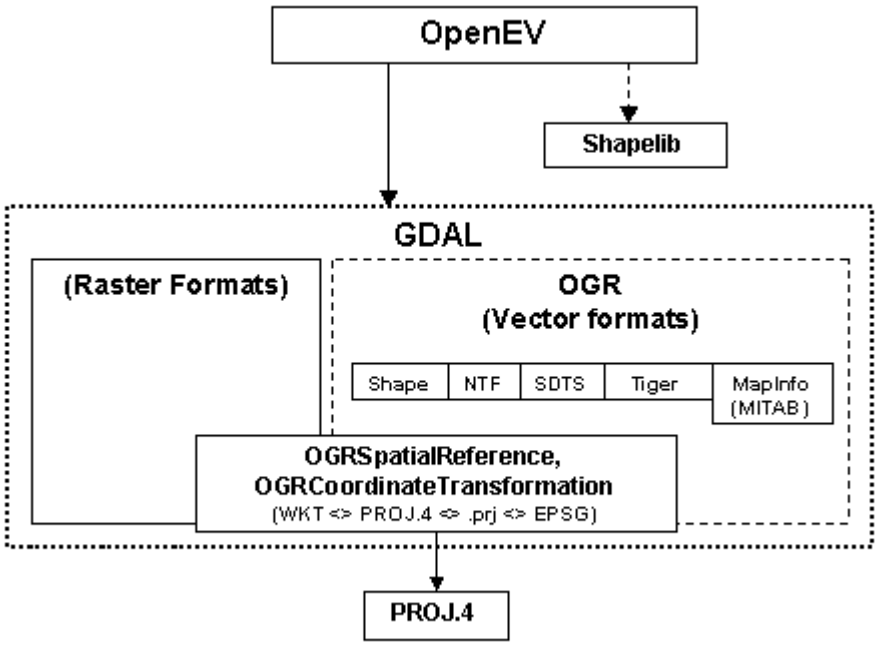

Fig. 4.1 Diagrama de las componentes y bibliotecas de OpenEV

4.3. Python

Python es un lenguaje de programación dinámico orientado a objetos que puede ser usado por una gran cantidad de tipos de software de desarrollo, y el cual se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation.

Python permite la reutilización de módulos en diversos programas. Además se incluye una colección de módulos estándar que se pueden utilizar como base de los programas, y módulos que proporcionan E/S de ficheros, llamadas a sistema, sockets y hasta interfaces a GUI.

Python es un lenguaje interpretado, lo que evita tener que compilar o enlazar ahorrando un tiempo considerable. El intérprete permite utilizarlo de modo interactivo, siendo útil para experimentar con programas desechables o probar funciones durante el desarrollo del programa.

Python es un lenguaje de programación multiparadigma, es decir, permite diversos estilos como: programación orientada a objetos, programación estructurada, programación funcional y programación orientada a aspectos. Además soporta otros muchos paradigmas mediante el uso de extensiones. Python posee resolución dinámica de nombres, lo que enlaza un método y un nombre de variable durante la ejecución del programa.

Otra característica del lenguaje es la facilidad de extensión, lo que permite que nuevos módulos se puedan escribir fácilmente en C o C++. Python puede utilizarse como un lenguaje de extensión para módulos y aplicaciones que necesitan de una interfaz programable.

4.4. CPython

CPython es la implementación del lenguaje de programación Python más extensamente usada y está escrito en C.

CPython es un intérprete de bytecode que tiene una interfaz para utilizar funciones externas con diferentes lenguajes, incluyendo el C, el cual debe escribir explícitamente bindigns en cualquier lenguaje a parte de Python.

En este trabajo Cpython ha sido utilizada para elaborar y testear los códigos de los algoritmos de los filtros CS, WMMR-MED y Volterra.

CAPÍTULO 5. RESULTADOS DE LAS PRUEBAS DE FILTRADO

Alcanzado el punto en el que los módulos están correctamente implementados y sin errores, llega el momento de comprobar si al realizar un filtrado los resultados son los esperados. Para ello se realizan diversos filtrados de imagen, modificando los parámetros en cada uno de ellos para tratar de saber en qué rango de los parámetros el filtro es más efectivo, y si corresponde con el resultado teórico.

Para ello se elige la imagen de la (**Fig. 5.1**), la cual corresponde a una imagen del fondo marino de las costas de Almería captada por un sónar de barrido lateral. La imagen ha sido proporcionada por la Unidad de Tecnología Marítima (CSIC).

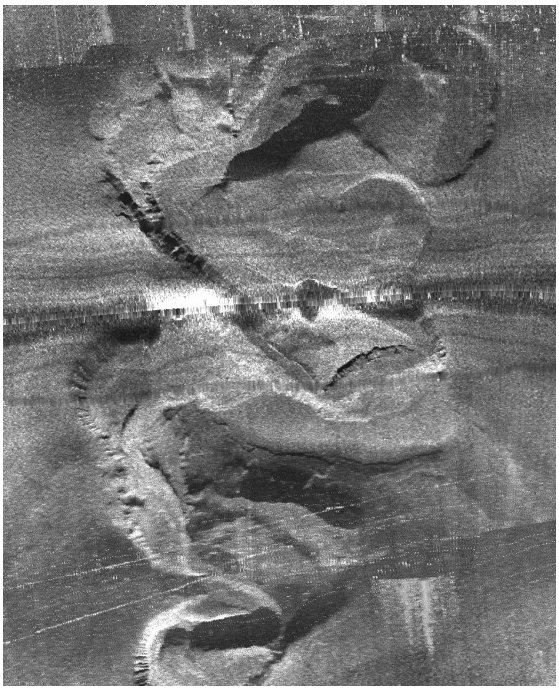

Fig. 5.1 Imagen del fondo marino de las costas de Almería.

Una vez filtrada la imagen, se verifica el efecto del filtrado sobre ella y se compara con otras imágenes pasadas por el mismo filtro pero con diferentes parámetros.

5.1. Filtro CS

Para la comparación de imágenes pasadas por el filtro CS se han tomado como muestras los parámetros siguientes:

- Tamaño de ventana: 3x3, 7x7 y 11x11.

- Parámetro A: 2, *N* / 2 +1 y *N*. Para 3x3: 2, 3, 5; para 7x7: 2, 13, 25 y para 11x11: 2, 31 y 61.

Los resultados obtenidos mediante el filtrado CS muestran que cuanto mayor es el valor de tamaño de ventana se produce un suavizado más pronunciado, además de aumentar el realce y definición de los bordes.

Si el valor del parámetro es elevado (a partir de 11x11) se empiezan a perder pequeños detalles de la imagen, situación que se agrava de forma importante para valores del parámetro A pequeños (**Fig. 5.2 (c)**).

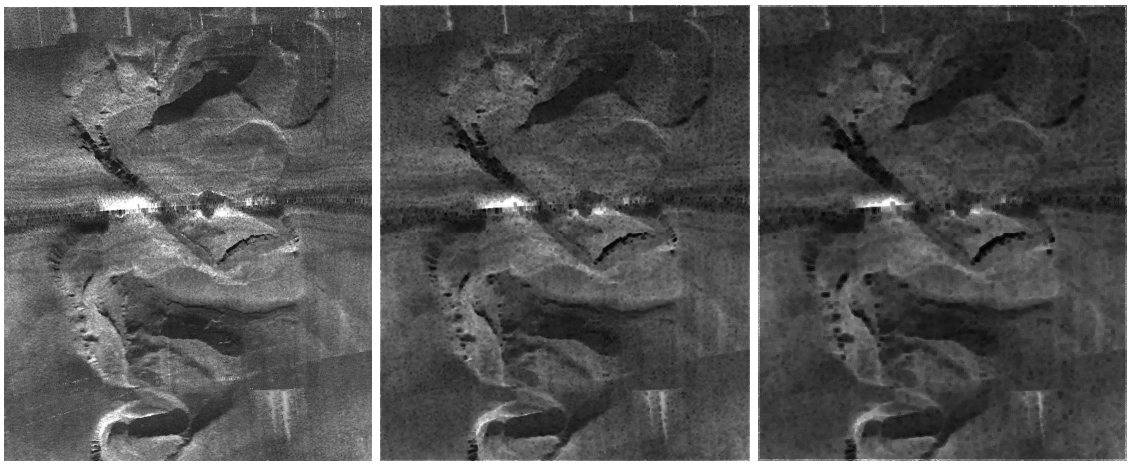

Fig. 5.2 Imágenes filtradas con filtro CS. Parámetro A:2. **(a)** Ventana: 3x3. **(b)** Ventana 7x7. **(c)** Ventana 11x11

En las imágenes de (**Fig. 5.2**), se comprueba que para valores del parámetro A pequeños, el filtro CS se comporta como un filtro mediana. Los efectos visibles son una buena eliminación de ruido y unos bordes más realzados, que se penalizan con una mala definición de los mismos y una importante pérdida de detalles. También se aprecia que en esta situación, a mayor valor del tamaño de ventana peor es la calidad visual de la imagen y menor intensidad general tiene.

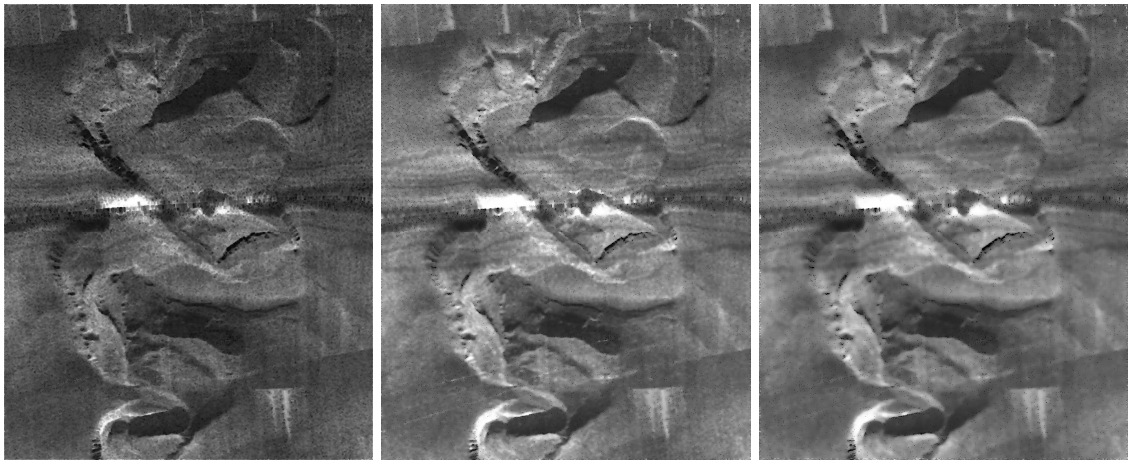

Fig. 5.3 Imágenes filtradas con filtro CS. **(a)** Ventana: 3x3. Parámetro A: 5. **(b)** Ventana 7x7, Parámetro A: 25. **(c)** Ventana 11x11, Parámetro A: 61

En las imágenes de la (**Fig. 5.3**) se tiene un valor de parámetro A elevado, es decir, los valores salientes del filtro corresponden a los más extremos del rango ordenado de muestras de la ventana deslizante. Esto produce un menor grado de eliminación del ruido que con valores del parámetro A menores, así como un menor realce de la imagen que el conseguido para valores medios del parámetro. Se produce un efecto difuminado que empeora en cierto grado la calidad de la imagen, además de un incremento de la intensidad general y de un suavizado que causa la pérdida de pequeños detalles.

Así pues, en consonancia con los resultados se observa que para obtener un buen realce de la imagen, así como una buena supresión del ruido la mejor elección es un valor medio del parámetro A y un tamaño de ventana entre 5x5 y 11x11 (dependiendo del grado de realce que se quiera obtener). Las imágenes obtenidas con estos parámetros se muestran en la (**Fig. 5.4**).

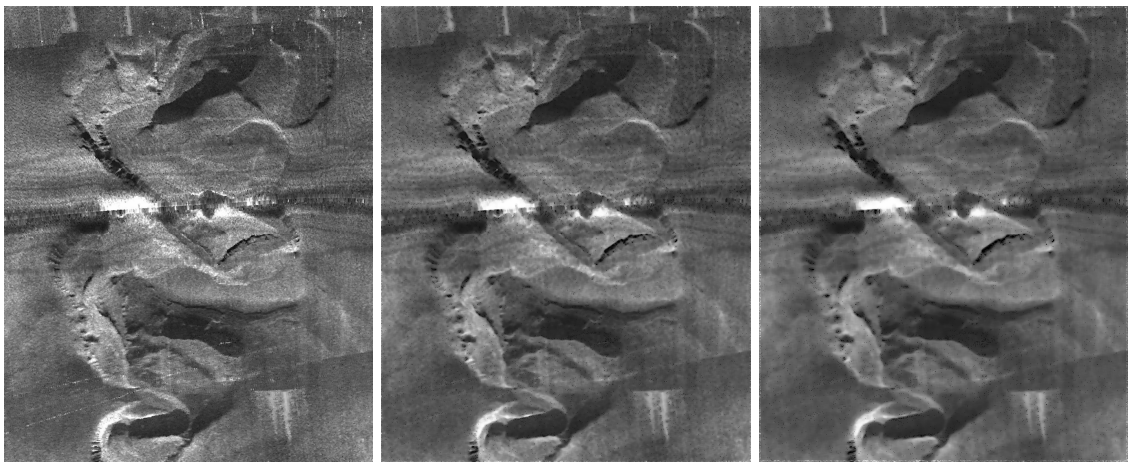

Fig. 5.4 Imágenes filtradas con filtro CS. **(a)** Ventana: 3x3. Parámetro A: 3. **(b)** Ventana 7x7, Parámetro A: 13. **(c)** Ventana 11x11, Parámetro A: 31

5.2. Filtro WMMR-MED

Para realizar la comparación de imágenes pasadas por el filtro WMMR-MED se han elegido los parámetros siguientes:

- Tamaño de ventana: 3x3, 7x7 y 11x11.
- Parámetro A: 0.2, 0.6 y 1.

En las imágenes obtenidas se observa que a mayor tamaño de ventana mayor es el suavizado de las zonas correspondientes a terrenos homogéneos, y se aprecia una leve reducción del nivel de ruido. Para grandes tamaños de ventana la imagen se difumina, a pesar de ello mantiene unos bordes bastante bien definidos.

En (**Fig. 5.5**) se muestran imágenes filtradas con un valor de parámetro A pequeño. En este caso se producen pocas modificaciones en la imagen, sólo se aprecia un poco de realce que aumenta cuanto mayor es el tamaño de ventana.

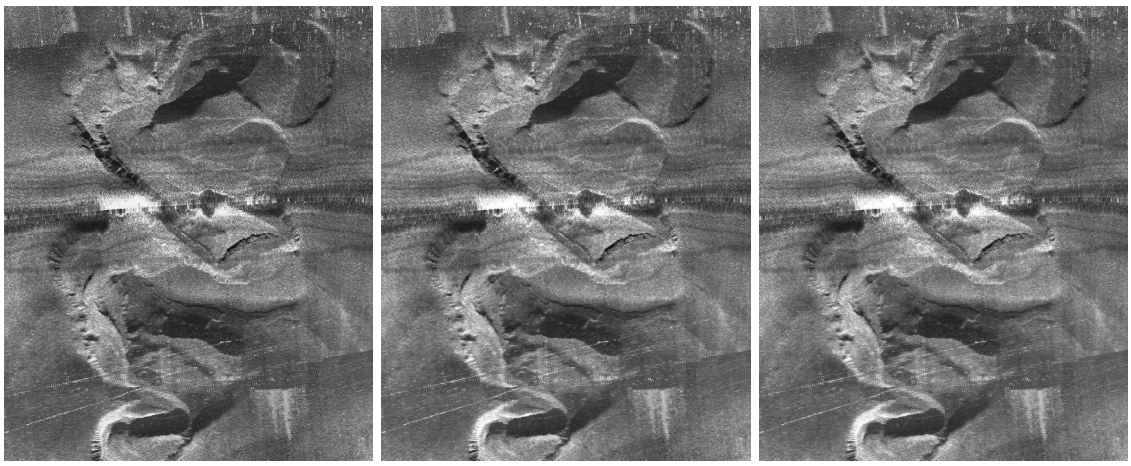

Fig. 5.5. Imágenes filtradas con filtro WMMR-MED. Parámetro A: 0.2. **(a)** Ventana: 3x3. **(b)** Ventana 7x7. **(c)** Ventana 11x11

Si se coge un valor de parámetro A elevado se consigue un buen grado de realce de la imagen, se observa una leve reducción del nivel de ruido y un suavizado de las zonas más homogéneas. Si se opta simultáneamente por un tamaño de ventana grande, se aprecia un progresivo aumento del difuminado de la imagen en consonancia con el valor del parámetro, apreciándose en gran medida en la (**Fig. 5.6 (c)**). Además se observa un incremento de la intensidad general de las imágenes de Fig. E a medida que se aumenta el tamaño de ventana, así como un mayor suavizado de las zonas homogéneas y una leve pérdida de definición de bordes y detalles.

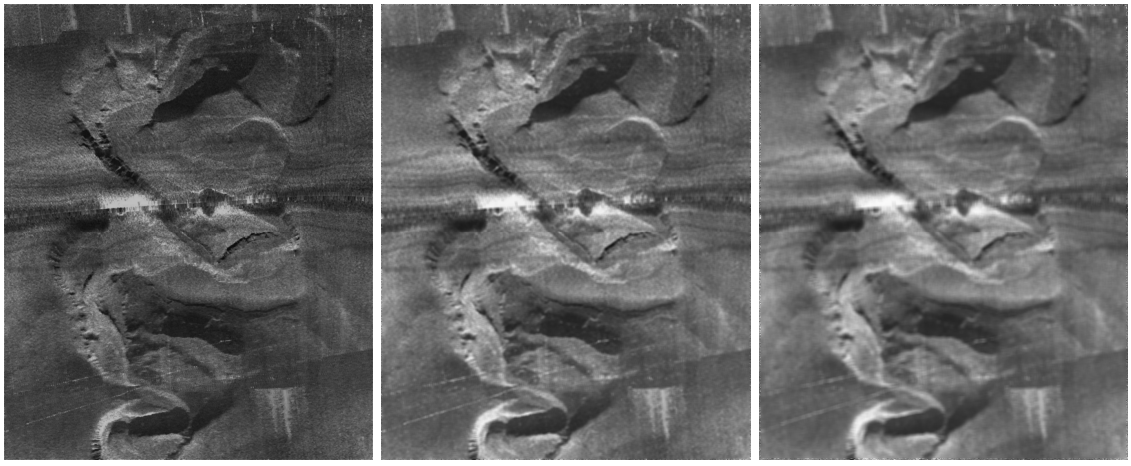

Fig. 5.6 Imágenes filtradas con filtro WMMR-MED. Parámetro A: 1. **(a)** Ventana: 3x3. **(b)** Ventana 7x7. **(c)** Ventana 11x11

Para valores medios del parámetro A se consigue un buen realce y definición de la imagen, se observa una reducción del nivel de ruido aceptable y un suavizado de las zonas más homogéneas. Si el tamaño de ventana es grande también aparece un poco de difuminado en la imagen, aunque mantiene un buen aspecto visual general y una buena definición de los bordes. En las imágenes de la (**Fig. 5.7**) se aprecia como se incrementa levemente la intensidad general al tratar con valores de tamaño de ventana grandes.

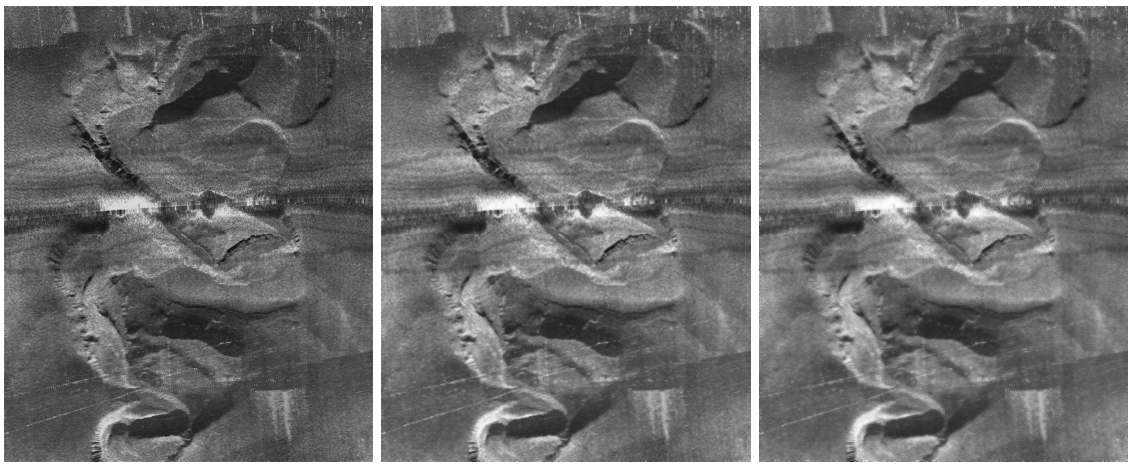

Fig. 5.7 Imágenes filtradas con filtro WMMR-MED. Parámetro A: 0.6. **(a)** Ventana: 3x3. **(b)** Ventana 7x7. **(c)** Ventana 11x11

A tenor de los resultados, se puede decir que para conseguir un óptimo realce de la imagen sin perder por ello pequeños detalles o definición de los bordes, la elección más adecuada es la de unos valores medios tanto del parámetro A como del tamaño de ventana.

5.3. Filtro Volterra

En ese apartado, para las imágenes pasadas por el filtro Volterra se han tomado los siguientes parámetros:

- Tamaño de ventana: 3x3, 7x7 y 11x11.
- Parámetro A: 0.005, 0.025 y 0.05.

En las imágenes resultantes del filtrado Volterra se observa como el valor del tamaño de la ventana modifica levemente el grado de realce de la imagen y de eliminación del ruido.

En (**Fig. 5.8**) se muestran imágenes filtradas con un valor de parámetro A pequeño. En este caso se producen pocas modificaciones en la imagen, aunque sí se aprecia un leve realce y reducción del ruido.

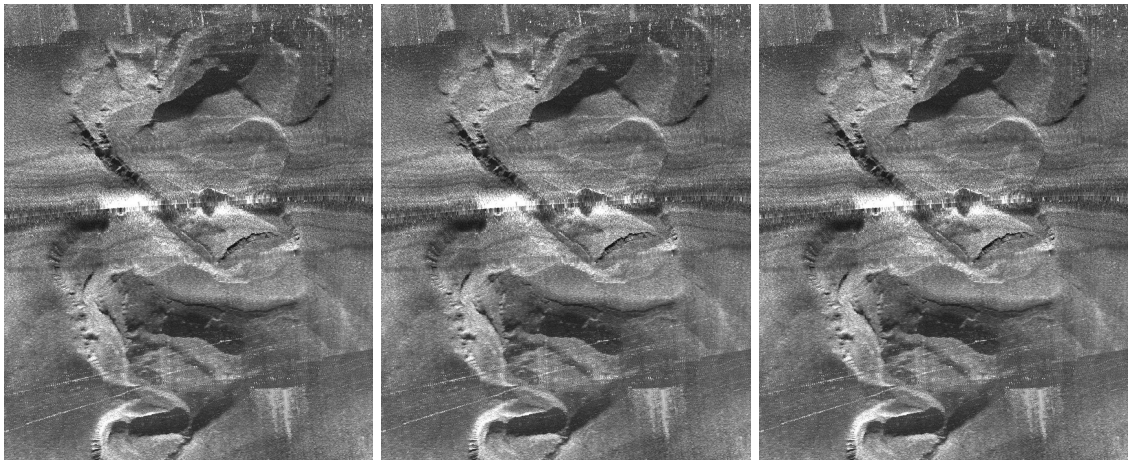

Fig. 5.8 Imágenes filtradas con filtro Volterra. Parámetro A: 0.005. **(a)** Ventana: 3x3. **(b)** Ventana 7x7. **(c)** Ventana 11x11

Si se opta por un valor del parámetro A cercano al 0.05 se observa un pequeño aumento del realce de la imagen y un descenso del nivel de ruido, situación mostrada en las imágenes de (**Fig. 5.9**).

Sin embargo, para valores mayores de 0.05 aparece ruido provocado por el exceso de realce aplicado a la imagen, por lo que es recomendable no superar dicho valor.

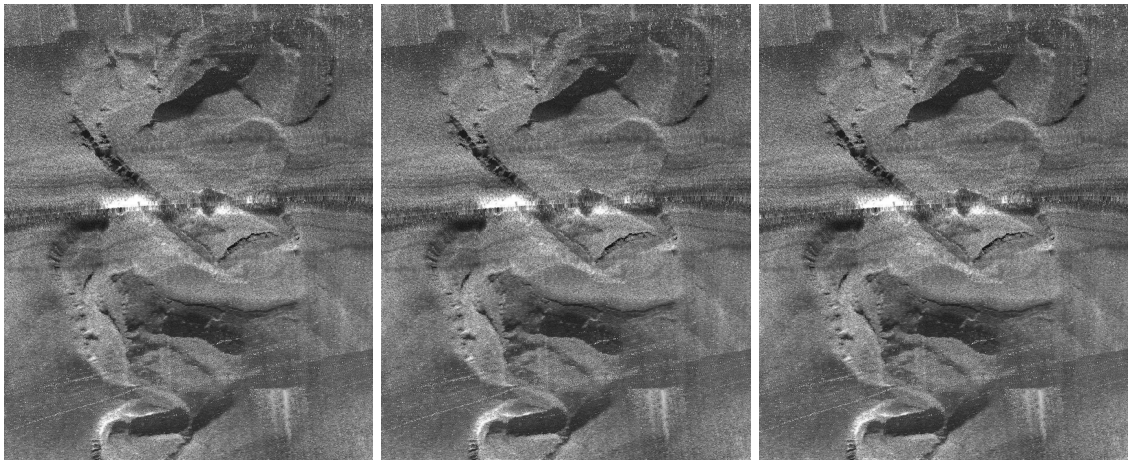

Fig. 5.9 Imágenes filtradas con filtro Volterra. Parámetro A: 0.05. **(a)** Ventana: 3x3. **(b)** Ventana 7x7. **(c)** Ventana 11x11

Por último, en (**Fig. 5.10**) se presentan imágenes con un valor del parámetro A de 0.025, elección que le proporciona una alto grado de eliminación de ruido, así como un buen realce y definición de los bordes. También se observa un suavizado en las zonas correspondientes a superficies homogéneas, mejorando de esta forma la calidad visual de la imagen.

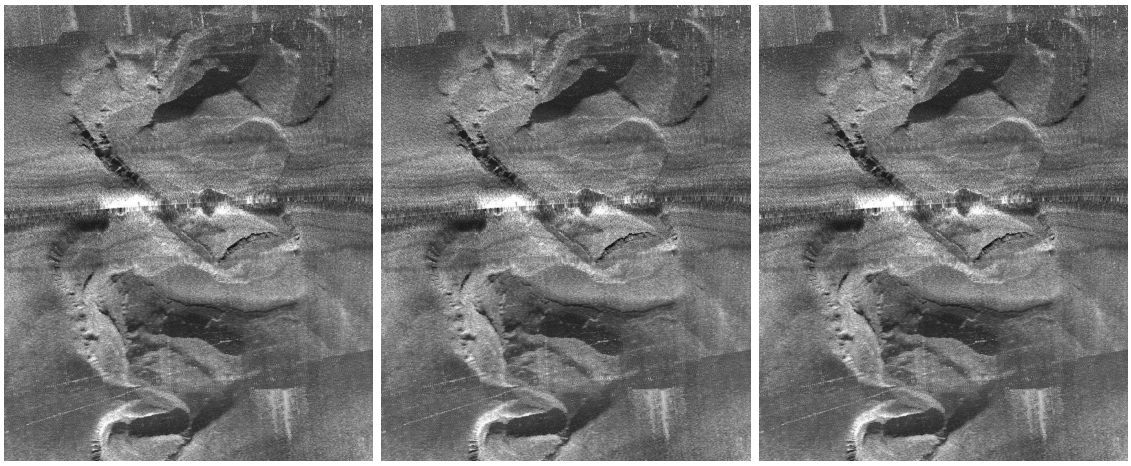

Fig. 5.10 Imágenes filtradas con filtro Volterra. Parámetro A: 0.025. **(a)** Ventana: 3x3. **(b)** Ventana 7x7. **(c)** Ventana 11x11

En conclusión, el filtro Volterra consigue mejores resultados en lo que respecta a realce y definición de la imagen al hacer uso de valores cercanos al 0.025.

CAPÍTULO 6. CONCLUSIONES

En este capítulo se presentan las conclusiones generales extraídas una vez finalizado el trabajo. Se presentan los siguientes apartados: objetivos iniciales del proyecto, desarrollo del trabajo, objetivos futuros, impacto medioambiental y conclusiones personales

6.1 Objetivos

Los objetivos iniciales del proyecto consistían en:

- Buscar y analizar información sobre los filtros CS, WMMR-MED y Volterra.
- Implementar los algoritmos de los filtros en Python.
- Implementar los módulos e integrarlos en la aplicación OpenEV.

Al término de este trabajo se han visto alcanzados los objetivos propuestos inicialmente.

6.2 Desarrollo del trabajo

La elaboración de este trabajo tiene como objetivo la implementación de una serie de módulos para la aplicación OpenEV, un software de código abierto, enfocados al filtrado espacial de imágenes captadas por sónar. Para su realización el proceso se divide en varias etapas, cada una con un objetivo concreto.

La primera etapa se centra en la búsqueda de información sobre el procesado de imágenes, tipos de filtrado y sus características, y el uso que se le da a cada uno de ellos. Así mismo también se analizan programas similares a OpenEV, como sería el caso de Grass y TNTlite. Estas aplicaciones ya implementan los filtros que se analizan en este trabajo, por lo que son de gran ayuda a la hora de testear los resultados obtenidos con los filtros propios. Al tratarse de software de código propietario, únicamente se puede extraer de ellos una comparación de los resultados obtenidos a iguales condiciones de filtrado. Simultáneamente se inicia el estudio de Python [21], el lenguaje de programación con el que se implementan los códigos de los filtros.

En una segunda etapa se produce el estudio de los tres filtros de realce espaciales a tratar y la implementación de sus algoritmos. Se analiza la documentación ofrecida en [20], referente a los filtros CS, WMMR y Volterra. Los artículos científicos obtenidos con esta búsqueda sirven como punto de partida para la comprensión de cada uno de los filtros y como ayuda en la búsqueda de información pertinente.

Una vez recogida y analizada suficiente información, se implementa el algoritmo de cada filtro y se testean en el software Cpython. Los resultados obtenidos con dichos algoritmos, los códigos de los cuales se pueden encontrar en Anexo C, se comparan a los que ofrece la aplicación TNTlite a mismas condiciones de filtrado para observar si el filtrado resultante es el correcto.

Una vez comprobado el funcionamiento de los códigos se procede al estudio de GTK+ y se analizan módulos ya existentes del software OpenEV, lo cual sirve de guía a la hora de organizar las funciones de los propios módulos.

Se implementa una interfaz de usuario común a los tres filtros y se comprueba su funcionamiento. Debido a que el software Cpython no soporta GTK+, a partir de este punto la implementación del código y sus testeos se realizan forzosamente en el mismo OpenEV. Las incompatibilidades entre ambos obligan a tener que modificar el código “al aire” en la propia aplicación, y a comprobar su funcionamiento cada vez que se realice un cambio en él.

Posteriormente se personaliza dicha interfaz para cada uno de los filtros, adoptando los valores predeterminados que ofrece la aplicación TNTlite. Se añade al módulo de cada filtro el código del algoritmo modificado para su correcta integración, y se hace uso del mayor número posible de métodos de la librería GDAL, dotando así de robustez y eficiencia al código. En (**Fig. 6.1**). se muestra la apariencia de OpenEV una vez integrados los tres módulos en la aplicación.

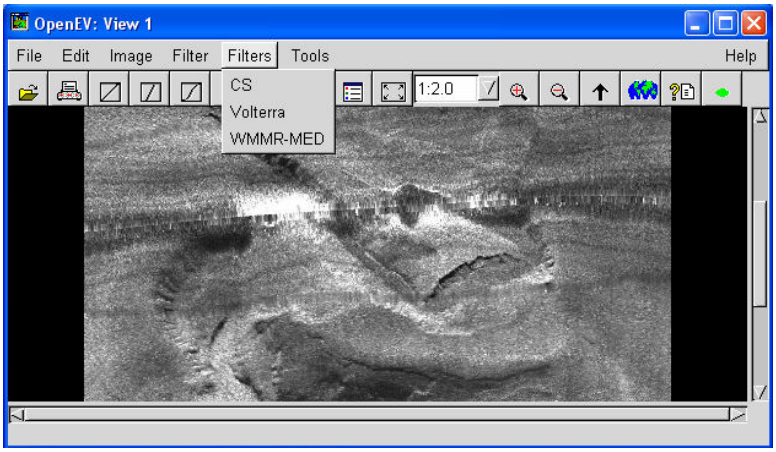

Fig. 6.1 OpenEV con los módulos de los filtros integrados

El proceso de integración del algoritmo inicial ha resultado complicado debido a problemas de incompatibilidad entre las versiones de Python existentes en Cpython y OpenEV. Algunos de los métodos usados en los filtros implementados en el software Cpython originan errores de ejecución en OpenEV, ya sean críticos o leves. En el caso de un error leve, al enviar la orden de filtrado no se produce respuesta por parte de la interfaz de usuario del módulo, y si se trata de un error crítico, simplemente el módulo con dicho error no es arrancado por la aplicación.

OpenEV no emite mensajes de error, por lo que no es posible averiguar de dónde procede el error que causa dicho fallo. Esta situación ha provocado que ésta haya sido la parte más costosa en la implementación de los módulos.

Una vez los errores se han eliminado se debe comprobar que los filtros actúan del modo esperado. Para ello se filtran imágenes captadas por sónar modificando los valores de los parámetros en cada prueba. El resultado es un abanico de imágenes que cubre de forma aproximada todo el rango de los parámetros, para posteriormente analizar con qué valores de los parámetros se obtienen mejores resultados y comparar los resultados obtenidos con las imágenes filtradas mediante TNTlite para los mismos filtros y parámetros.

6.3 Futuros objetivos

El trabajo desarrollado se puede tomar como base para una futura ampliación en cuanto al número de filtros espaciales disponibles para la aplicación, en concreto de filtros de realce.

Una extensión de los módulos aquí presentados podría ser una ampliación de las opciones que ofrecen los módulos en los siguientes campos:

- *Tipo de ventana deslizante:* Incluir algún otro tipo a parte de la cuadrada, como los presentados en el apartado 2.5.

- *Tamaño de ventana:* Ofrecer un tamaño de ventana configurable por el propio usuario.

- *Información:* Mostrar al usuario información relacionada con el proceso de filtrado, tiempo de ejecución, coste computacional, etc.

Una ampliación más general sería la implementación de módulos con filtros de características diferentes, incrementando el abanico de posibilidades para el procesado de rasters de datos geoespaciales que ofrece OpenEV.

6.4 Impacto medioambiental

El estudio del impacto medioambiental es necesario en todo proyecto por mínimo que este pudiera parecer. Unas consecuencias perjudiciales para el medioambiente pueden truncar el desarrollo o implantación de un proyecto.

El desarrollo y distribución de unos módulos destinados a ampliar una aplicación software de código abierto para el procesado de rasters y/o vectores de datos geoespaciales, a priori, no tiene consecuencias adversas severas. Tanto en la etapa de desarrollo, como en la de distribución y uso del software, los recursos empleados son principalmente equipos informáticos. La distribución del software a través de Internet evita el uso de dispositivos físicos

de almacenamiento, e implícitamente, supone un ahorro en recursos materiales y energía.

Por otro lado, la utilización de los filtros implementados en este trabajo puede proporcionar un aumento de la información extraída de las imágenes de sónar. Una información mejor aproximada a la situación real puede favorecer el aumento de la seguridad en las rutas marítimas, ayudar a la protección de ecosistemas marinos o facilitar la localización de objetos y accidentes del fondo marino, por ejemplo.

6.5 Conclusiones personales

La realización de este trabajo me ha proporcionado nuevos conocimientos en un área anteriormente desconocida para mí, el procesado de imágenes digitales. Además, la duración que requería el mismo me ha permitido mejorar la metodología de trabajo y la organización a llevar durante su desarrollo.

La posible agregación en el futuro de los módulos implementados a la aplicación OpenEV es un atractivo objetivo a cumplir, ya que se trata de colaborar en el desarrollo de software distribuido bajo la licencia GNU LGPL.

En conclusión, ha resultado un trabajo estimulante ya sea tanto por su desarrollo y proceso previo de documentación, como por la implementación final de las técnicas de realce.

BIBLIOGRAFÍA

Libros

[1] Jensen, J. R., “Image enhancement”, Cap. 7 en *Introductory digital image processing: A remote sensing perspective*, pp. 139-195, Prentice-Hall, UpperSaddle River N.J., (2004).

[2] Gonzalez, R. C., *Digital image processing*, pp. 34-161, Prentice-Hall, UpperSaddle River N.J., (2002).

[3] Rabiee, H. R. and Kashyap, R. L., “Pre-processing and post-processing of images and image sequences”, Cap. 6 en *Adaptive multiresolution image and video compression and pre/post-processing of image and video streams*, pp. 77-95, Purdue Libraries, West Lafayette, Indiana, (1996).

Artículos de revistas

[4] Yong Lee and Adly Fam, “An edge gradient enhancing adaptive order statistic filter”, *Acoustics, Speech, and Signal Processing, IEEE Transactions on* 35 (5), 680-695 (1987).

[5] Hardie, R.C. and Boncelet, C., “LUM filters: a class of rank-order-based filters for smoothing and sharpening”, *Signal Processing, IEEE Transactions on* 41 (3), 1061-1076 (1993).

[6] Thurnhofer, S. and Mitra, S.K., “A general framework for quadratic Volterra filters for edge enhancement”, *Image Processing, IEEE Transactions on* 5 (6), 950-963 (1996).

[7] Mitra, S.K., Li, H., Lin, I.-S. and Yu, T.-H., “A new class of nonlinear filters for image enhancement”, *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on* 4, 2525-2528 (1991).

[8] Hardie, R.C. and Boncelet, C.G., “Gradient-based edge detection using nonlinear edge enhancing prefilters”, *Image Processing, IEEE Transactions on* 4 (11), 1572-1577 (1995).

[9] Longbotham, H. and Eberly, D., “The WMMR filters: a class of robust edge enhancers”, *Signal Processing, IEEE Transactions on* 41 (4), 1680-1685 (1993).

[10] Hardie, R.C. and Barner, K.E., “Extended permutation filters and their application to edge enhancement”, *Image Processing, IEEE Transactions on* 5 (6), 855-867 (1996).

[11] Sintes, C. and Solaiman, B., “Side scan sonar and interferometric noise”, *Microwaves, Radar and Wireless Communications. 2000. MIKON-2000. 13th International Conference on* 1, 371 – 375 (2000).

[12] Taiho Koh and Powers, E., “Second-order Volterra filtering and its application to nonlinear system identification”, *Acoustics, Speech, and Signal Processing, IEEE Transactions on* 33 (6), 1445-1455 (1985).

[13] Sicuranza, G.L., “Quadratic filters for signal processing”, *Proceedings of the IEEE* 80 (8), 1263-1285 (1992).

[14] Longbotham, H.G., Bovik, A.C. and Restrepo, A., “Generalized order statistic filters“, *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on* 3, 1610-1613 (1989).

[15] Maussang, F., Chanussot, J., Hétet, A. and Amate, M., “Higher-Order Statistics for the Detection of Small Objects in a Noisy Background Application on Sonar Imaging”, *Applied Signal Processing, EURASIP Journal on* 2007 (1), 25-25 (2007).

[16] Bell, J.M. and Linnett, L.M., “Simulation and analysis of synthetic sidescan sonar images”, *Radar, Sonar and Navigation, IEE Proceedings* 144 (4), 219-226 (1997).

[17] Blondel, Ph., Parson, L.M. and Robigou, V., “TexAn: textural analysis of sidescan sonar imagery and generic seafloor characterisation”, *OCEANS '98 Conference Proceedings* 1 (28), 419-423 (1998).

[18] He, Q. and Zhang, Z., “A new edge detection algorithm for image corrupted by White-Gaussian noise”, *International Journal of Electronics and Communications* 61, 546-550 (2007).

[19] Mansourpour, M., Rajabi, M. A. and Blais, J. A. R., “Effects and performance of speckle noise reduction filters on active radar and SAR images”, *WG I/5 & I/6 Workshop on Topographic Mapping from Space*, (2006).

Recursos web

[20] Tutorial de la aplicación TNTlite, URL <http://www.microimages.com/i18n/_es_spanish/es_filter.pdf>

[21] Manual de Python, URL <http://marmota.act.uji.es/MTP/pdf/python.pdf>

[22] Página oficial del lenguaje Python, URL <http://www.python.org/>

[23] Documento teoría sónar, “Influencia del estudio de la ecuación de sónar en la operación de las unidades submarinas”, URL <http://www.armada.mil.ve/comnaop/escuadra/submarino/textos/40_Aniversario/Resumen%20Teor%C3%ADa%20de%20Sonar.rtf>

[24] Documento teoría sónar, URL <http://www.topografiaglobal.com.ar/archivos/teoria/sonar.html>

[25] Manual de Hidrografía, pp. 5-14, URL <http://www.iho.shom.fr/publicat/free/files/Man_Hid_cap_4.pdf>

[26] Documento sobre la reducción del ruido grano en imágenes SAR, URL <*http://www.mappinginteractivo.com/plantilla-ante.asp?id_articulo=1111*>

[27] Documento sobre reducción del ruido en imágenes digitales, “Imágenes de ultrasonido”, URL <*http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2005/ultrasonido/Imagenes%20de%20ultrasonido.pdf*>

[28] Documento sobre el ruido grano, URL <http://dukemil.egr.duke.edu/Ultrasound/k-space/node5.html>

[29] Documento sobre el ruido en imágenes digitales, “Procesamiento digital de señales ultrasónicas en end”, pp.1-10, URL <http://www.iai.csic.es/ritul/PubCartagena/Luis/pdsend.doc>

[30] Página oficial del software TNTlite, URL <http://www.microimages.com/tntlite/>

[31] Información sobre Cpython en la web Wikipedia, URL <http://en.wikipedia.org/wiki/CPython>

[32] Página oficial de la distribución FWTools, URL <http://fwtools.maptools.org/>

[33] Página oficial del software Grass, URL <http://grass.itc.it/>

ANEXOS

TÍTULO DEL TFC: Técnicas de procesado de datos de sónar de barrido lateral

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Carlos Agreda Ninot

DIRECTOR: Claudio Lo lacono

FECHA: 28 de noviembre de 2007

ANEXO A. DEFINICIONES

En este anexo se presentan dos conceptos diferentes que por espacio y conveniencia no se han incluido en el cuerpo del trabajo.

En el primer apartado se presenta brevemente el concepto de borde de una imagen y la técnica de procesado de detección de bordes.

En el apartado A.2 se define la Ley de Weber-Fechner.

A.1. Detección de bordes

Se define borde como cualquier discontinuidad que sufre alguna función de intensidad sobre los puntos de la misma.
Existen diferentes tipos de bordes en función de:

- Cambio brusco en la distancia transductor-objeto
- Cambio en la normal del objeto
- Cambio en la reflectancia del objeto
- Cambio en la proyección de la luz incidente

La detección de bordes es uno de los procesados más significativos para imágenes digitales ya que a partir de él se puede extraer gran cantidad de información.

A partir de la información extraída de los bordes de una imagen se puede saber donde se sitúa el objeto escaneado, su forma, su tamaño, así como su textura. Los bordes se encuentran en zonas de una imagen donde el nivel de intensidad fluctúa bruscamente, marcándose más cuando el cambio de intensidad es más acentuado.

En la detección de los bordes de una imagen es vital encontrar los puntos bordes que los forman, que son dos o más píxeles adyacentes con una gran diferencia de valores de intensidad.

A.2. Ley de Weber-Fechner

La ley de Weber-Fechner establece que *'el menor cambio discernible en la magnitud de un estímulo es proporcional a la magnitud del estímulo'*, estableciendo una relación cuantitativa entre la magnitud del estímulo físico y como éste es percibido.

Aplicado al procesado de imágenes digitales, la ley caracteriza la percepción de las diferencias en el nivel de gris en relación a la intensidad general de la

imagen. Por lo tanto, la percepción de los detalles de la imagen es proporcional a la media de la intensidad de los píxeles vecinos. Esto hace que los detalles se aprecien más fácilmente en regiones oscuras, mientras que las claras tienden a enmascararlos.

A pesar de que la ley de Weber-Fechner no describe dependencias para valores de intensidad pequeños, se puede utilizar la proporcionalidad como una estimación global.

ANEXO B. DOMINIO ESPACIAL Y FRECUENCIAL

A continuación se presentan el dominio espacial y el frecuencial, así como una breve definición del teorema de convolución.

2.1. Dominio espacial

El término dominio espacial se refiere al conjunto de píxeles que componen una imagen, y los métodos en el dominio espacial son procedimientos que operan directamente sobre los píxeles. Las funciones de procesamiento se pueden expresar como

$$g(x,y) = T\,[f(x,y)] \qquad \textbf{(2.1)}$$

donde $f(x,y)$ es la imagen de entrada, $g(x,y)$ es la imagen procesada y T es un operador que actúa sobre f, definido en algún entorno de (x,y).

La aproximación usada para definir un entorno alrededor de (x,y) es emplear un área de subimagen cuadrada centrada en (x,y). El centro de la subimagen se mueve píxel a píxel comenzando en la esquina superior izquierda y aplicando el operador T en cada posición (x,y) para obtener g.

Por lo tanto, el filtrado en el dominio del espacio consiste en determinar g en un punto (x,y) a partir de los valores de f en un entorno predefinido (x,y).

2.2. Dominio frecuencial

Las técnicas en el dominio de la frecuencia se basan en el teorema de convolución. Sea $g(x,y)$ una imagen formada por la convolución de una imagen $f(x,y)$ y un operador lineal invariante de posición $h(x,y)$, es decir

$$g(x,y) = h(x,y) * f(x,y). \qquad \textbf{(2.2)}$$

Entonces. Por el teorema de convolución, se cumple la siguiente relación en el dominio de la frecuencia:

$$G(u,v) = H(u,v)F(u,v) \qquad \textbf{(2.3)}$$

donde G, H y F son respectivamente las transformadas de Fourier de g, h y f.

En una aplicación típica de mejora de la imagen, $f(x,y)$ es conocida y el objetivo, después de calcular $F(u,v)$, es seleccionar $H(u,v)$ de forma que la imagen deseada,

$$g(x,y) = F^{-1}[H(u,v)F(u,v)] \quad \textbf{(2.4)}$$

presente resaltada alguna característica de $g(x,y)$, y donde F^{-1} es la transformada inversa de Fourier.

2.2.1. Teorema de convolución

La convolución de dos funciones $f(x)$ y $g(x)$, indicada por $f(x)*g(x)$, se define mediante la integral

$$f(x)*g(x) = \int_{-\infty}^{\infty} f(\alpha)g(x-\alpha)d\alpha \quad \textbf{(2.5)}$$

donde α es una variable ficticia para la integración.

Se escribe **(2.5)** de forma distinta

$$F[(f*g)(x)] = F[f(x)]\ F\ [g(x)]. \quad \textbf{(2.6)}$$

ANEXO C. CÓDIGOS

En el anexo C, se muestran los códigos desarrollados durante el trabajo.

En el apartado C.1., se presentan los códigos de los algoritmos implementados en la aplicación Cpython.

De la misma forma, en el apartado C.2., se presentan los códigos de los módulos implementados para la aplicación OpenEV.

C.1. Códigos de los algoritmos de los filtros CS, WMMR-MED y Volterra

En los siguientes subapartados se muestran los códigos de los algoritmos implementados en la aplicación Cpython.

C.1.1. Filtro CS

```
# Filtro Realce
# Filtro CS (Comparación y selección)
# Kernel 3x3,5x5,7x7,9x9,11x11 (ventana convolución)

# Módulos importados
from Numeric import *
from PIL import Image
import gdal
import gdalnumeric

# Métodos definidos
# Crea un objeto GDAL Dataset object asociado con un array
# asociándole la proyección y georeferencia, y escribiendo
# un fichero de salida
def SaveArrayWithGeo(array,src_filename,dst_filename,format='GTiff'):
  # Lee la info del archivo fuente
  src_ds = gdal.Open(src_filename)
  gt = src_ds.GetGeoTransform()
  pj = src_ds.GetProjection()
  src_ds = None
  # Crea GDAL dataset para array y establece georeferencia
  src_ds = gdalnumeric.OpenArray(array)
  src_ds.SetGeoTransform(gt)
  src_ds.SetProjection(pj)
  # Escribe array dataset en un nuevo archivo
  driver = gdal.GetDriverByName(format)
  if driver is None:
    raise ValueError,"No se pudo encontrar driver "+format
  return driver.CreateCopy(dst_filename,src_ds)

# Definición función ventana convolución
def ventanaConv(nker,array,h,w):
  ker = [0]*(nker**2)
  for i in range(h-((nker-1)/2),h+((nker-1)/2)+1):
    for j in range(w-((nker-1)/2),w+((nker-1)/2)+1):
      ker.append(array[i][j])
```

```
  return ker

# Definición función mean (media aritmética)
def mean(data):
  tot=0.0
  for i in range(len(data)):
    tot += data[i]
  return tot/len(data)

# Programa principal
nombre = raw_input('Introduce el nombre del archivo: ')
try:
  # Open File with GDAL
  imgIn = gdal.Open(nombre)
  src = gdalnumeric.DatasetReadAsArray(imgIn)
  dst = gdalnumeric.DatasetReadAsArray(imgIn)
  w = imgIn.RasterXSize
  h = imgIn.RasterYSize
  nker = int(raw_input('Introduce el tamaño de la ventana de
convolución (3,5,7,9,11):'))
  while(nker!=3 and nker!=5 and nker!=7 and nker!=9 and nker!=11):
    nker = int(raw_input('Introduce el tamaño de la ventana de
convolución (3,5,7,9,11):'))
  # Ventana convolución (matriz)
  matriz = nker**2
  param = int(raw_input('Introduce parámetro A: '))
  while(param<=0 or param>((matriz+1)/2)):
    param = int(raw_input('Introduce parámetro A: '))
  # Los bordes no se modifican
  for i in range(((nker-1)/2),h-((nker-1)/2)):
    for j in range(((nker-1)/2),w-((nker-1)/2)):
      ker = ventanaConv(nker,src,i,j)
      med = mean(ker)
      ker.sort()
      min = ker[param-1]
      max = ker[len(ker)-1-param]
      if(src[i][j]<med):
        pixelOut = min
      else:
        pixelOut = max
      dst[i][j] = pixelOut
      print dst[i][j]   #usado sólo para comprobar píxeles salientes
  # Convertir array en imagen con infoGeo
  SaveArrayWithGeo(dst,nombre,'filtradaCSprueba5x5.tif')
  print('Foto guardada con éxito')

except:
  print('No existe dicho fichero o la ruta es incorrecta.')
```

C.1.2. Filtro WMMR-MED

```
# Filtro Realce
# Filtro WMMR
# Kernel 3x3,5x5,7x7,9x9,11x11 (ventana convolución)

# Módulos importados
from Numeric import *
from PIL import Image
import gdal
```

```
import gdalnumeric

# Métodos definidos
# Crea un objeto GDAL Dataset object asociado con un array
# asociándole la proyección y georeferencia, y escribiendo
# un fichero de salida
def SaveArrayWithGeo(array,src_filename,dst_filename,format='GTiff'):
  # Lee la info del archivo fuente
  src_ds = gdal.Open(src_filename)
  gt = src_ds.GetGeoTransform()
  pj = src_ds.GetProjection()
  src_ds = None
  # Crea GDAL dataset para array y establece georeferencia
  src_ds = gdalnumeric.OpenArray(array)
  src_ds.SetGeoTransform(gt)
  src_ds.SetProjection(pj)
  # Escribe array dataset en un nuevo archivo
  driver = gdal.GetDriverByName(format)
  if driver is None:
    raise ValueError,"No se pudo encontrar driver "+format
  return driver.CreateCopy(dst_filename,src_ds)

# Definición función ventana convolución
def ventanaConv(nker,array,h,w):
  ker = [0]*(nker**2)
  for i in range(h-((nker-1)/2),h+((nker-1)/2)+1):
    for j in range(w-((nker-1)/2),w+((nker-1)/2)+1):
      ker.append(array[i][j])
  return ker

# Definición función mean (media aritmética)
def mean(data):
  tot=0.0
  for i in range(len(data)):
    tot += data[i]
  return tot/len(data)

# Definición función WMMR
def wmmr(data):
  med = (len(data)-1)/2
  for i in range(med):
    weight = data[med+i] - data[i]
    if(i==0 or weight<min):
      min = weight
      num = i
  ker = data[i:(med+i)]

  return mean(ker)

# Programa principal
nombre = raw_input('Introduce el nombre del archivo: ')
try:
  # Open File with GDAL
  imgIn = gdal.Open(nombre)
  src = gdalnumeric.DatasetReadAsArray(imgIn)
  dst = gdalnumeric.DatasetReadAsArray(imgIn)
  w = imgIn.RasterXSize
  h = imgIn.RasterYSize
  nker = int(raw_input('Introduce el tamaño de la ventana de
convolución (3,5,7,9,11):'))
  while(nker!=3 and nker!=5 and nker!=7 and nker!=9 and nker!=11):
```

```
    nker = int(raw_input('Introduce el tamaño de la ventana de
convolución (3,5,7,9,11):'))
  # Ventana convolución (matriz)
  matriz = nker**2
  param = float(raw_input('Introduce parámetro A (entre 0 y 1): '))
  while(param<0 or param>1):
    param = float(raw_input('Introduce parámetro A: '))
  # Los bordes no se modifican
  for i in range(((nker-1)/2),h-((nker-1)/2)):
    for j in range(((nker-1)/2),w-((nker-1)/2)):
      ker = ventanaConv(nker,src,i,j)
      ker.sort()
      value = wmmr(ker)
      dst[i][j] = value*param + src[i][j]*(1-param)
      print dst[i][j]   #usado sólo para comprobar píxeles salientes
  # Convertir array en imagen con infoGeo
  SaveArrayWithGeo(dst,nombre,'filtradaWMMR2.tif')
  print('Foto guardada con éxito')

except:
  print('No existe dicho fichero o la ruta es incorrecta.')
```

C.1.3. Filtro Volterra

```
# Filtro Realce
# Filtro Volterra
# Kernel 3x3,5x5,7x7,9x9,11x11 (ventana convolución)

# Módulos importados
from Numeric import *
from PIL import Image
import gdal
import gdalnumeric

# Métodos definidos
# Crea un objeto GDAL Dataset object asociado con un array
# asociándole la proyección y georeferencia a él, y escribiendo
# un fichero de salida
def SaveArrayWithGeo(array,src_filename,dst_filename,format='GTiff'):
  # Lee la info del archivo fuente
  src_ds = gdal.Open(src_filename)
  gt = src_ds.GetGeoTransform()
  pj = src_ds.GetProjection()
  src_ds = None
  # Crea GDAL dataset para array y establece georeferencia
  src_ds = gdalnumeric.OpenArray(array)
  src_ds.SetGeoTransform(gt)
  src_ds.SetProjection(pj)
  # Escribe array dataset en un nuevo archivo
  driver = gdal.GetDriverByName(format)
  if driver is None:
    raise ValueError,"No se pudo encontrar driver "+format
  return driver.CreateCopy(dst_filename,src_ds)

# Definición función filtro Volterra Cuadrático
def volterra(array,i,j,param):
  term1 = param*3*(array[i][j]**2)
  term2 = param*0.5*array[i+1][j+1]*array[i-1][j-1]
  term3 = param*0.5*array[i+1][j-1]*array[i-1][j+1]
```

```
  term4 = param*array[i+1][j]*array[i-1][j]
  term5 = param*array[i][j+1]*array[i][j-1]
  pix = term1-term2-term3-term4-term5

  #El siguiente conjunto de operaciones no funciona en la aplicación
#Cpython
  #pix = param*(3*(array[i][j]**2)-(array[i+1][j+1]*array[i-1][j-
1])/2-  (array[i+1][j-1]*array[i-1][j+1])/2-array[i+1][j]*array[i-
1][j]- array[i][j+1]*array[i][j-1])
  return pix

# Programa principal
nombre = raw_input('Introduce el nombre del archivo: ')
try:
  # Open File with GDAL

  imgIn = gdal.Open(nombre)
  src = gdalnumeric.DatasetReadAsArray(imgIn)
  dst = gdalnumeric.DatasetReadAsArray(imgIn)
  w = imgIn.RasterXSize
  h = imgIn.RasterYSize
  # Ventana convolución (matriz)
  nker = 3
  matriz = nker**2
  param = float(raw_input('Introduce parámetro A: '))
  while(param<=0 or param>((matriz+1)/2)):
    param = float(raw_input('Introduce parámetro A: '))
  # Los bordes no se modifican
  for i in range(((nker-1)/2),h-((nker-1)/2)):
    for j in range(((nker-1)/2),w-((nker-1)/2)):
      edge = volterra(src,i,j,param)
      if ((src[i][j]+edge)>255):
        pixOut = 255
      else:
        pixOut = src[i][j] + edge
      dst[i][j] = pixOut

# Convertir array en imagen con infoGeo
  SaveArrayWithGeo(dst,nombre,'filtradaVolterraTNT.tif')
  print('Foto guardada con éxito')

except:
  print('No existe dicho fichero o la ruta es incorrecta.')
```

C.2. Códigos de los módulos correspondientes a los filtros CS, WMMR-MED y Volterra

A continuación se presentan los códigos de los módulos implementados para la aplicación OpenEV.

C.2.1. Módulo del filtro CS

```
# OpenEV module filter_cs.py

from gtk import *
```

```
import gview
import GtkExtra
import gviewapp
import gdalnumeric
import Numeric


# Definicion funcion ventana convolucion
def conv_win( array, nker, h, w):
    ker = [0]*(nker**2)
    h_start = h-((nker-1)/2)
    h_end = h+((nker-1)/2)+1
    w_start = w-((nker-1)/2)
    w_end = w+((nker-1)/2)+1

    for i in range(h_start, h_end):
        for j in range(w_start, w_end):
            ker.append(array[i][j])
    return ker


# Definicion funcion mean (media aritmetica)
def mean( data ):
    tot = 0.0
    length = len(data)
    for i in range(length):
        tot += data[i]
    return tot/length


def get_raster_size(layer):
    w = layer.get_parent().get_dataset().GetRasterBand(1).XSize
    h = layer.get_parent().get_dataset().GetRasterBand(1).YSize
    return (w,h)


def layer_is_raster(layer):
    """returns TRUE if layer is raster and FALSE if else"""
    try:
        layer.get_nodata(0)
        return TRUE
    except:
        return FALSE

##########################################################################
###

class FilterCSTool(gviewapp.Tool_GViewApp):

    def __init__(self,app=None):
        gviewapp.Tool_GViewApp.__init__(self,app)
        self.init_menu()

    def launch_dialog(self,*args):
        self.win = FilterCSDialog()
        self.win.show()

    def init_menu(self):
        self.menu_entries.set_entry("Filters/CS",2,self.launch_dialog)
```

```
######################################################################
###

class FilterCSDialog(GtkWindow):
    def scale_upper(self, adj):
        nker = float(self.adj1.value)
        max = (nker**2 + 1)/2
        value = float(self.adj2.value)
        self.adj2.set_all(value, 1.0, max, 1.0, 2.0, 0.0)

    def __init__(self,app=None):
        GtkWindow.__init__(self)
        self.set_title('Filter CS')
        self.create_gui()
        self.show()

    def show(self):
        GtkWindow.show_all(self)

    def close(self, *args):
        self.destroy()

    def create_gui(self):
        self.main_box = GtkVBox(spacing = 10)
        self.main_box.set_border_width(10)
        self.add(self.main_box)

        self.frame = GtkFrame("Parameters")
        self.main_box.add(self.frame)
        self.frame.show()

        self.vbox = GtkVBox()
        self.vbox.set_border_width(5)
        self.frame.add(self.vbox)

#### PARAMETERS
#########################################################
        self.hbox = GtkHBox()
        self.vbox.pack_start(self.hbox, True, True, 5)

        self.vbox1 = GtkVBox()
        self.hbox.pack_start(self.vbox1, False, True, 5)

        self.label = GtkLabel('Kernel size :')
        self.label.set_alignment(0, 0.5)
        self.vbox1.pack_start(self.label, True, True, 0)

        self.adj1 = GtkAdjustment(3.0, 3.0, 15.0, 2.0, 4.0, 0.0)
        self.spinner_ker = GtkSpinButton(self.adj1, 0, 0)
        self.spinner_ker.set_update_policy(UPDATE_ALWAYS)
        self.vbox1.pack_start(self.spinner_ker, FALSE, FALSE,0)

        self.vbox1 = GtkVBox()
        self.hbox.pack_start(self.vbox1, True, True, 5)

        self.label = GtkLabel('Parameter A :')
        self.label.set_alignment(0, 0.5)
        self.vbox1.pack_start(self.label, True, True, 0)

        nker = float(self.adj1.value)
        max = (nker**2 + 1)/2
```

```
        self.adj2 = GtkAdjustment(2.0, 1.0, max, 1.0, 2.0, 0.0)
        self.hscale = GtkHScale(self.adj2)
        self.adj2.connect("value_changed", self.scale_upper)
        self.hscale.set_update_policy(UPDATE_DELAYED)
        self.hscale.set_digits(0)
        self.vbox1.pack_start(self.hscale, FALSE, FALSE,0)

#### BUTTONS
############################################################
        self.hbox = GtkHBox(True, 10)
        self.main_box.pack_start(self.hbox, False, True, 5)

        self.button = GtkButton('Ok')
        self.button.connect("clicked", self.execute_filter)
        self.hbox.pack_start(self.button, FALSE, True,0)

        self.button = GtkButton('Cancel')
        self.button.connect("clicked", self.close)
        self.hbox.pack_start(self.button, FALSE, True,0)

        self.main_box.show()

#########################################################################
###

    def execute_filter( self, *args ):
        self.layer = gview.app.sel_manager.get_active_layer()
        if not layer_is_raster(self.layer):
            return None
        (w,h) = get_raster_size(self.layer)
        ds = self.layer.get_parent().get_dataset()
        src = gdalnumeric.DatasetReadAsArray(ds)
        dst = gdalnumeric.DatasetReadAsArray(ds)
        nker = int(self.spinner_ker.get_value())
        param = int(self.adj2.value)

        # Los bordes no se modifican
        nker_start = ((nker-1)/2)
        h_end = h - nker_start
        w_end = w - nker_start

        for i in range(nker_start, h_end):
            for j in range(nker_start, w_end):
                ker = conv_win(src, nker, i, j)
                med = mean(ker)
                sker = Numeric.sort(ker)
                if((i==2) and (j==2)):
                    print sker
                n = (len(sker)-1)/2
                min = sker[n+1-param]
                max = sker[n+1+param]

                if src[i][j] < med:
                    dst[i][j] = min
                else:
                    dst[i][j] = max

        array_name = gdalnumeric.GetArrayFilename(dst)

        gview.app.file_open_by_name(array_name)
```

```
TOOL_LIST = ['FilterCSTool']
```

C.2.2. Módulo del filtro WMMR-MED

```
# OpenEV module filter_wmmr.py

from gtk import *

import gview
import GtkExtra
import gviewapp
import gdalnumeric
import Numeric


# Definicion funcion ventana convolucion
def conv_win( array, nker, h, w):
    ker = [0]*(nker**2)
    h_start = h-((nker-1)/2)
    h_end = h+((nker-1)/2)+1
    w_start = w-((nker-1)/2)
    w_end = w+((nker-1)/2)+1

    for i in range(h_start, h_end):
        for j in range(w_start, w_end):
            ker.append(array[i][j])
    return ker


# Definicion funcion mean (media aritmetica)
def mean(data):
    tot = 0.0
    length = len(data)
    for i in range(length):
        tot += data[i]
    return tot/length


# Definicion funcion WMMR
def wmmr(data):
    med = (len(data)-1)/2
    for i in range(med):
        weight = data[med+i] - data[i]
        if((i==0) or (weight<min)):
            min = weight
            num = i
    ker = data[i:(med+i)]
    return mean(ker)


def get_raster_size(layer):
    w = layer.get_parent().get_dataset().GetRasterBand(1).XSize
    h = layer.get_parent().get_dataset().GetRasterBand(1).YSize
    return (w,h)
```

```
def layer_is_raster(layer):
    """returns TRUE if layer is raster and FALSE if else"""
    try:
        layer.get_nodata(0)
        return TRUE
    except:
        return FALSE

#########################################################################
###

class FilterWMMRTool(gviewapp.Tool_GViewApp):

    def __init__(self,app=None):
        gviewapp.Tool_GViewApp.__init__(self,app)
        self.init_menu()

    def launch_dialog(self,*args):
        self.win = FilterWMMRDialog()
        self.win.show()

    def init_menu(self):
        self.menu_entries.set_entry("Filters/WMMR-
MED",2,self.launch_dialog)

#########################################################################
###

class FilterWMMRDialog(GtkWindow):

    def __init__(self,app=None):
        GtkWindow.__init__(self)
        self.set_title('Filter WMMR')
        self.create_gui()
        self.show()

    def show(self):
        GtkWindow.show_all(self)

    def close(self, *args):
        self.destroy()

    def create_gui(self):
        self.main_box = GtkVBox(spacing = 10)
        self.main_box.set_border_width(10)
        self.add(self.main_box)

        self.frame = GtkFrame("Parameters")
        self.main_box.add(self.frame)
        self.frame.show()

        self.vbox = GtkVBox()
        self.vbox.set_border_width(5)
        self.frame.add(self.vbox)

#### PARAMETERS
#########################################################
        self.hbox = GtkHBox()
        self.vbox.pack_start(self.hbox, True, True, 5)

        self.vbox1 = GtkVBox()
```

```
        self.hbox.pack_start(self.vbox1, False, True, 5)

        self.label = GtkLabel('Kernel size :')
        self.label.set_alignment(0, 0.5)
        self.vbox1.pack_start(self.label, True, True, 0)

        self.adj1 = GtkAdjustment(3.0, 3.0, 15.0, 2.0, 4.0, 0.0)
        self.spinner_ker = GtkSpinButton(self.adj1, 0, 0)
        self.spinner_ker.set_update_policy(UPDATE_ALWAYS)
        self.vbox1.pack_start(self.spinner_ker, FALSE, FALSE,0)

        self.vbox1 = GtkVBox()
        self.hbox.pack_start(self.vbox1, True, True, 5)

        self.label = GtkLabel('Parameter A :')
        self.label.set_alignment(0, 0.5)
        self.vbox1.pack_start(self.label, True, True, 0)

        self.adj2 = GtkAdjustment(0.2, 0.0, 1.0, 0.01, 0.01, 0.0)
        self.hscale = GtkHScale(self.adj2)
        self.hscale.set_update_policy(UPDATE_DELAYED)
        self.hscale.set_digits(2)
        self.vbox1.pack_start(self.hscale, FALSE, FALSE,0)

#### BUTTONS
############################################################
        self.hbox = GtkHBox(True, 10)
        self.main_box.pack_start(self.hbox, False, True, 5)

        self.button = GtkButton('Ok')
        self.button.connect("clicked", self.execute_filter)
        self.hbox.pack_start(self.button, FALSE, True,0)

        self.button = GtkButton('Cancel')
        self.button.connect("clicked", self.close)
        self.hbox.pack_start(self.button, FALSE, True,0)

        self.main_box.show()

########################################################################
###

    def execute_filter( self, *args ):
        self.layer = gview.app.sel_manager.get_active_layer()
        if not layer_is_raster(self.layer):
            return None
        (w,h) = get_raster_size(self.layer)
        ds = self.layer.get_parent().get_dataset()
        src = gdalnumeric.DatasetReadAsArray(ds)
        dst = gdalnumeric.DatasetReadAsArray(ds)
        nker = int(self.spinner_ker.get_value())
        param = float(self.adj2.value)

        # Los bordes no se modifican
        nker_start = ((nker-1)/2)
        h_end = h - nker_start
        w_end = w - nker_start

        for i in range(nker_start, h_end):
            for j in range(nker_start, w_end):
                ker = conv_win(src, nker, i, j)
```

```
                sker = Numeric.sort(ker)
                if((i==2) and (j==2)):
                    print sker
                value = wmmr(sker)
                dst[i][j] = value*(1-param) + src[i][j]*param

        array_name = gdalnumeric.GetArrayFilename(dst)
)
        gview.app.file_open_by_name(array_name)


TOOL_LIST = ['FilterWMMRTool']
```

C.2.3. Módulo del filtro Volterra

```
# OpenEV module filter_wmmr.py

from gtk import *

import gview
import GtkExtra
import gviewapp
import gdalnumeric
import Numeric
from PIL import Image


# Definicion funcion filtro Volterra Cuadratico
def volterra(array,i,j,param):
    div = 1.0/27
    return (div*param*(3*(array[i][j]**2) - (array[i+1][j+1]*array[i-
1][j-1])/2 \
           - (array[i+1][j-1]*array[i-1][j+1])/2 -
array[i+1][j]*array[i-1][j] \
           - array[i][j+1]*array[i][j-1]))

def get_raster_size(layer):
    w = layer.get_parent().get_dataset().GetRasterBand(1).XSize
    h = layer.get_parent().get_dataset().GetRasterBand(1).YSize
    return (w,h)


def layer_is_raster(layer):
    """returns TRUE if layer is raster and FALSE if else"""
    try:
        layer.get_nodata(0)
        return TRUE
    except:
        return FALSE

##################################################################
###

class FilterVolterraTool(gviewapp.Tool_GViewApp):

    def __init__(self,app=None):
        gviewapp.Tool_GViewApp.__init__(self,app)
```

```
        self.init_menu()

    def launch_dialog(self,*args):
        self.win = FilterVolterraDialog()
        self.win.show()

    def init_menu(self):

self.menu_entries.set_entry("Filters/Volterra",2,self.launch_dialog)

########################################################################
###

class FilterVolterraDialog(GtkWindow):

    def __init__(self,app=None):
        GtkWindow.__init__(self)
        self.set_title('Filter Volterra')
        self.create_gui()
        self.show()

    def show(self):
        GtkWindow.show_all(self)

    def close(self, *args):
        self.destroy()

    def create_gui(self):
        self.main_box = GtkVBox(spacing = 10)
        self.main_box.set_border_width(10)
        self.add(self.main_box)

        self.frame = GtkFrame("Parameters")
        self.main_box.add(self.frame)
        self.frame.show()

        self.vbox = GtkVBox()
        self.vbox.set_border_width(5)
        self.frame.add(self.vbox)

#### PARAMETERS
############################################################
        self.hbox = GtkHBox()
        self.vbox.pack_start(self.hbox, True, True, 5)

        self.vbox1 = GtkVBox()
        self.hbox.pack_start(self.vbox1, False, True, 5)

        self.label = GtkLabel('Kernel size :')
        self.label.set_alignment(0, 0.5)
        self.vbox1.pack_start(self.label, True, True, 0)

        self.adj1 = GtkAdjustment(3.0, 3.0, 15.0, 2.0, 4.0, 0.0)
        self.spinner_ker = GtkSpinButton(self.adj1, 0, 0)
        self.spinner_ker.set_update_policy(UPDATE_ALWAYS)
        self.vbox1.pack_start(self.spinner_ker, FALSE, FALSE,0)

        self.vbox1 = GtkVBox()
        self.hbox.pack_start(self.vbox1, True, True, 5)

        self.label = GtkLabel('Parameter A :')
```

```
        self.label.set_alignment(0, 0.5)
        self.vbox1.pack_start(self.label, True, True, 0)

        self.adj2 = GtkAdjustment(0.005, 0.0, 0.1, 0.001, 0.001, 0.0)
        self.hscale = GtkHScale(self.adj2)
        self.hscale.set_update_policy(UPDATE_DELAYED)
        self.hscale.set_digits(3)
        self.vbox1.pack_start(self.hscale, FALSE, FALSE,0)

#### BUTTONS
############################################################
        self.hbox = GtkHBox(True, 10)
        self.main_box.pack_start(self.hbox, False, True, 5)

        self.button = GtkButton('Ok')
        self.button.connect("clicked", self.execute_filter)
        self.hbox.pack_start(self.button, FALSE, True,0)

        self.button = GtkButton('Cancel')
        self.button.connect("clicked", self.close)
        self.hbox.pack_start(self.button, FALSE, True,0)

        self.main_box.show()

#########################################################################
###

    def execute_filter( self, *args ):
        self.layer = gview.app.sel_manager.get_active_layer()
        if not layer_is_raster(self.layer):
            return None
        (w,h) = get_raster_size(self.layer)
        ds = self.layer.get_parent().get_dataset()
        src = gdalnumeric.DatasetReadAsArray(ds)
        dst = gdalnumeric.DatasetReadAsArray(ds)
        nker = int(self.spinner_ker.get_value())
        param = float(self.adj2.value)

        # Los bordes no se modifican
        nker_start = ((nker-1)/2)
        h_end = h - nker_start
        w_end = w - nker_start

        for i in range(nker_start, h_end):
            for j in range(nker_start, w_end):
                edge = volterra(src, i, j, param)

                if(src[i][j] + edge) < 0:
                    dst[i][j] = src[i][j] + edge
                elif(src[i][j] + edge) > 255:
                    dst[i][j] = src[i][j] - edge
                else:
                    dst[i][j] = src[i][j]

        array_name = gdalnumeric.GetArrayFilename(dst)

        gview.app.file_open_by_name(array_name)


TOOL_LIST = ['FilterVolterraTool']
```